NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

Priority: IST-2004-2.4.7 – "Semantic-based knowledge and content systems"

D6.4.1 Realisation & early evaluation of NeOn service-oriented registry repository

Deliverable Co-ordinator:	**Walter Waterfeld**
Deliverable Co-ordinating Institution:	**Software AG (SAG)**
Other Authors:	**Raul Palma (UPM)**
Contributors:	**Peter Haase (UKarl)**

This document describes the prototype deliverable D6.4.1 NeOn service-oriented registry repository of the NeOn basic infrastructure layer.

Document Identifier:	NEON/2007/D6.4.1/v1.1.1	Date due:	October 31, 2007
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	December 15, 2007
Project start date:	March 1, 2006	Version:	V1.1.1
Project duration:	4 years	State:	Final
		Distribution:	Public

NeOn Consortium

This document is a part of the NeOn research project funded by the IST Programme of the Commission of the European Communities by the grant number IST-2005-027595. The following partners are involved in the project:

Open University (OU) – Coordinator Knowledge Media Institute – KMi Berrill Building, Walton Hall Milton Keynes, MK7 6AA United Kingdom Contact person: Martin Dzbor, Enrico Motta E-mail address: {m.dzbor, e.motta} @open.ac.uk	**Universität Karlsruhe – TH (UKARL)** Institut für Angewandte Informatik und Formale Beschreibungsverfahren – AIFB Englerstrasse 28 D-76128 Karlsruhe, Germany Contact person: Peter Haase E-mail address: pha@aifb.uni-karlsruhe.de
Universidad Politécnica de Madrid (UPM) Campus de Montegancedo 28660 Boadilla del Monte Spain Contact person: Asunción Gómez Pérez E-mail address: asun@fi.upm.es	**Software AG (SAG)** Uhlandstrasse 12 64297 Darmstadt Germany Contact person: Walter Waterfeld E-mail address: walter.waterfeld@softwareag.com
Intelligent Software Components S.A. (ISOCO) Calle de Pedro de Valdivia 10 28006 Madrid Spain Contact person: Jesús Contreras E-mail address: jcontreras@isoco.com	**Institut 'Jožef Stefan' (JSI)** Jamova 39 SI-1000 Ljubljana Slovenia Contact person: Marko Grobelnik E-mail address: marko.grobelnik@ijs.si
Institut National de Recherche en Informatique et en Automatique (INRIA) ZIRST – 655 avenue de l'Europe Montbonnot Saint Martin 38334 Saint-Ismier France Contact person: Jérôme Euzenat E-mail address: jerome.euzenat@inrialpes.fr	**University of Sheffield (USFD)** Dept. of Computer Science Regent Court 211 Portobello street S14DP Sheffield United Kingdom Contact person: Hamish Cunningham E-mail address: hamish@dcs.shef.ac.uk
Universität Koblenz-Landau (UKO-LD) Universitätsstrasse 1 56070 Koblenz Germany Contact person: Steffen Staab E-mail address: staab@uni-koblenz.de	**Consiglio Nazionale delle Ricerche (CNR)** Institute of cognitive sciences and technologies Via S. Martino della Battaglia, 44 - 00185 Roma-Lazio, Italy Contact person: Aldo Gangemi E-mail address: aldo.gangemi@istc.cnr.it
Ontoprise GmbH. (ONTO) Amalienbadstr. 36 (Raumfabrik 29) 76227 Karlsruhe Germany Contact person: Jürgen Angele E-mail address: angele@ontoprise.de	**Food and Agriculture Organization of the United Nations (FAO)** Viale delle Terme di Caracalla 1 00100 Rome Italy Contact person: Marta Iglesias E-mail address: marta.iglesias@fao.org
Atos Origin S.A. (ATOS) Calle de Albarracín, 25 28037 Madrid Spain Contact person: Tomás Pariente Lobo E-mail address: tomas.parientelobo@atosorigin.com	**Laboratorios KIN, S.A. (KIN)** C/Ciudad de Granada, 123 08018 Barcelona Spain Contact person: Antonio López E-mail address: alopez@kin.es

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed writing parts of this document:

Ontoprise GmbH

Change Log

Version	Date	Amended by	Changes
0.8	2007-11-26	Peter Haase	Evaluation
0.9	2007-11-27	Raul Palma	Streamlined Oyster Server part
1.0	2007-12-05	Walter Waterfeld	Incorporated review feedback
1.0.1	2007-12-06	Walter Waterfeld	Incorporated review feedback
1.1.0	2007-12-10	Walter Waterfeld	Incorporated Oyster Server parts
1.1.1	2007-12-12	Angela Wilkinson	Grammatical and spelling corrections

Executive Summary

This document describes the prototype deliverable D6.4.1 NeOn service-oriented registry repository of the NeOn basic infrastructure layer. It contains the definition of a NeOn registry service, which is based on the OMV ontology meta model. This is used to specialize the general purpose ebXML registry service to an ontology registry service. This service has been realized with the research oriented Oyster registry server as well as with the general purpose commercial SOA registry repository CentraSite. Additionally a repository service is provided, which is integrated with registry functionality. The available services have been evaluated according to the previously specified requirements.

Table of Contents

List of Figures

1 Introduction

This document describes the prototype deliverable D6.4.1 realisation & early evaluation of NeOn service-oriented registry repository.

Since the actual deliverable consists of software components, we restrict the presentation here to a brief overview of the work done and the resulting artefacts.

2 Ontology Registry

The purpose of an ontology registry is to gather information about ontologies. Thus in addition to data describing the ontology it contains a reference to the ontology but not the ontology itself. The ontology itself is stored in an ontology repository, which is described in the next chapter.

The Ontology registry functionality will be used in the NeOn toolkit by engineering components, which have to manage several ontologies. Thus they can query and analyse the registry information about them to process them accordingly.

At first we explain some important concepts, which influenced the design of the NeOn ontology registry service. We then describe the service interface. In the following we present two realisations of the registry service. Finally the positioning of these two realisations is discussed and an outlook of the planned usage of the NeOn registry service is sketched.

2.1 Conceptual Issues

We briefly discuss the two major concepts for the realisation of the NeOn registry services, which are the OMV ontology meta model and the general purpose ebXML registry standard.

As already specified in the NeOn architecture all major infrastructure functionality will be available as services in order to allow the loose coupling of components. For the NeOn registry functionality this clearly means to realize them as web services.

2.1.1 OMV

The Ontology Metadata Vocabulary OMV is a standard proposal for describing ontologies that was originally developed based on discussions and agreements within the EU IST thematic network of excellence Knowledge Web and that has been adopted (and further elaborated) within NeOn as the standard metadata for networked ontologies. The OMV metadata schema is formally represented as an ontology and is designed modularly: OMV distinguishes between the OMV Core and various OMV Extensions. The core captures information which is expected to be relevant to the majority of ontology reuse settings, while the extensions allow ontology developers and users to specify task/application-specific ontology-related information (e.g. mappings, ontology evaluation, ontology changes etc.). These extensions should be compatible to the OMV core schema but at the same time fulfil the requirements of a domain, task or community-driven setting. The OMV elements are classified according to their impact on the prospected reusability of the described ontology content (e.g. required, optional) and according to the type and purpose of the contained information (e.g. availability, provenance, applicability, etc.). For a complete reference and the complete ontology, we refer the reader to http://omv.ontoware.org/

The main classes and properties of the OMV core ontology are illustrated in Figure 2.1.1. Besides the main class Ontology, OMV also models additional classes and properties required to support the reuse of ontologies, especially in the context of the Semantic Web, such as Party, Organisation, Person, LicenseModel, OntologyLanguage, OntologySyntax and OntologyTask among others.

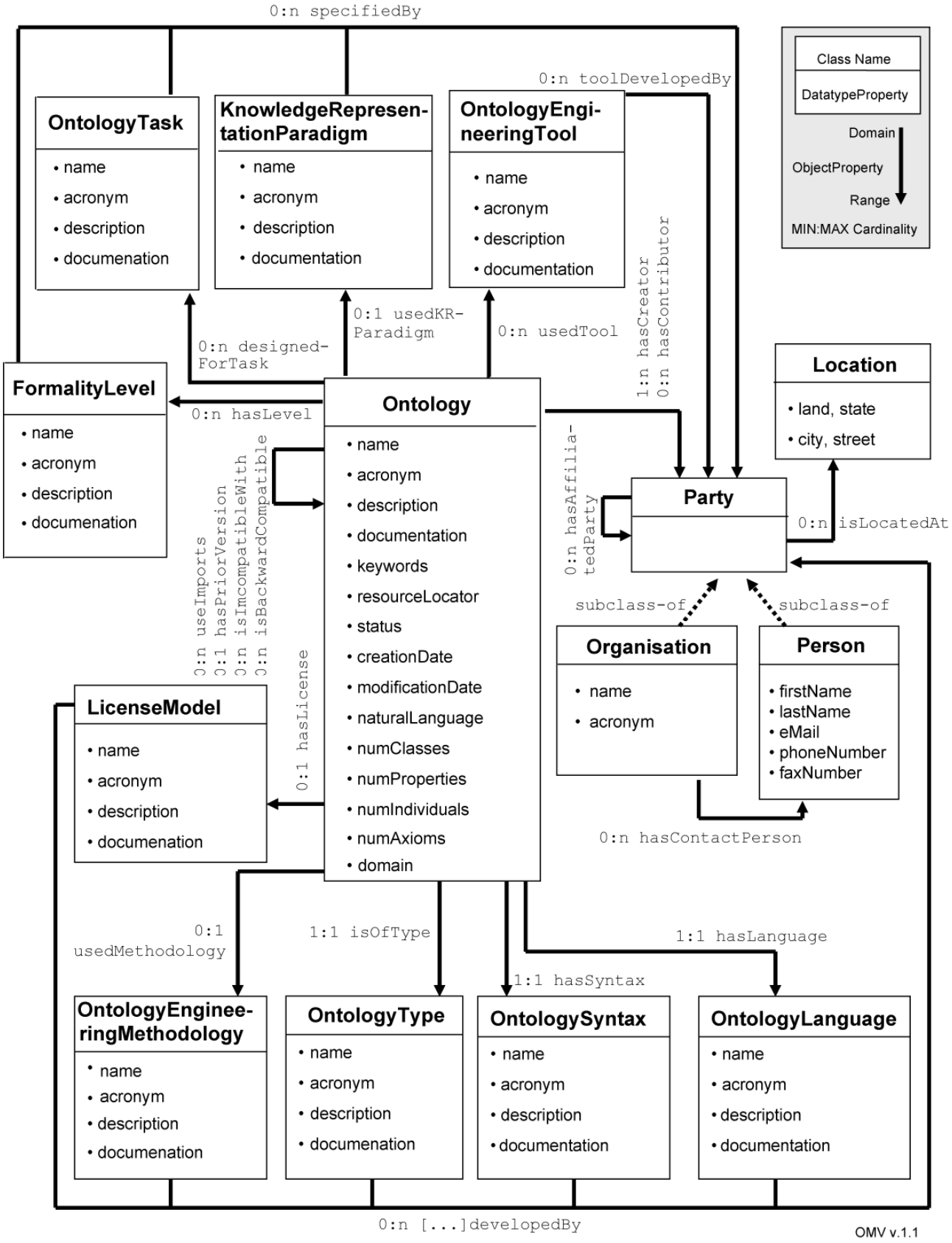


Figure 2.1.1. OMV core model

2.1.2 General Purpose Registry: ebXML registry

There are two major registry specifications in use. The most common one is certainly UDDI. It is especially tailored for web services and can hardly be used to describe other artefacts, as its data model is not extensible.

The other registry specification is ebXML registry [Fuger2006]. This has an extensible data model and thus can be adopted to register arbitrary artefacts. For this it provides an interface, which allows adding arbitrary classes. Instances of those classes can be created, updated and queried with a generic interface based on the class RegistryObject.

In general it covers broader registry functionality and especially contains the widely accepted UDDI functionality. The specification contains a general purpose registry information model (with namespace prefix rim), which is defined via an XML schema. The main interface for an ebXML registry is the ebXML registry services [FugerNajmi2006].

With JAXR [Najmi2002] there exists a common Java API for both registries. The data model of it is however very much oriented towards ebXML with some smaller extensions for UDDI.

The ebXML registry aims at a registry to manage standardized and extensible metadata of any domain. It explicitly can support security and federation. Additionally, it defines an integration of repository and registry functionality.

The main interface of an ebXML registry is defined via web services.

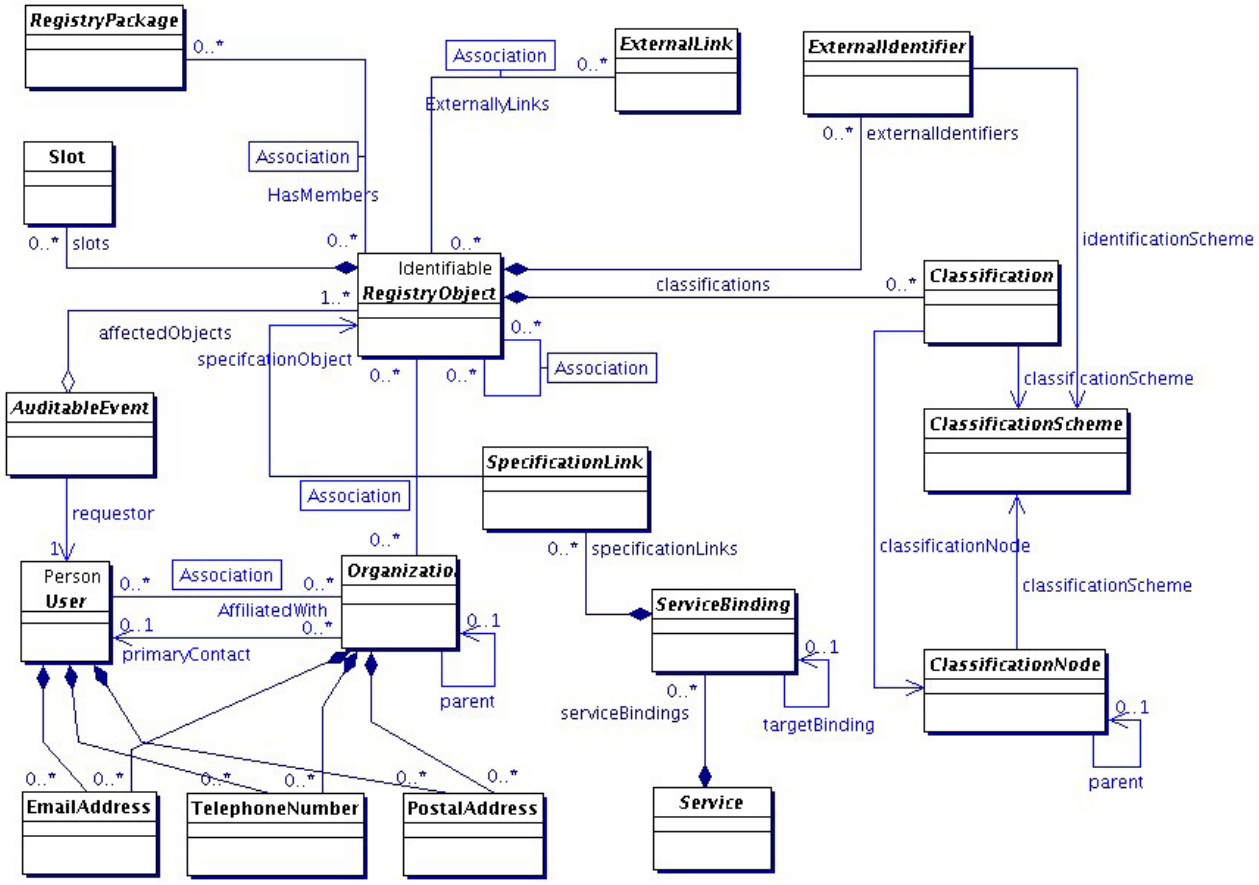


Figure 2.1.2. EbXML Registry Information Model (RIM)

2.1.3 OMV Specialized ebXML Registry

OMV has been independently defined as an ontology registry meta model. Nevertheless, there is a big overlap of properties with the general purpose registry ebXML, e.g.: identifier, name, description. That means for this OMV parts no change in ebXML is necessary.

All the other OMV classes will be added to the ebXML model. This would allow creating, updating and querying instances of those classes via the ebXML services as generic RegistryObjects. However applications with the NeOn registry interface will mainly deal with OMV objects. A generic ebXML interface would be quite difficult to handle.

Therefore we define a specialized interface just for OMV objects. ebXML registry has besides a general purpose registry interface already a specific part for the provenance model of ebXML (organization, user etc). This allows a more comfortable interface by directly using organization and user elements instead of dynamic general registry object elements.

Along the same lines we then specialize the ebXML registry web service interface for the OMV model. That means we can directly pass <rim_omv:Ontology> elements instead of <rim:RegistryObject objectType="Ontology">. This is done by defining an OMV specific schema with types for all OMV classes as extensions of the general purpose ebXML types. Additionally also the query interface is specialized to the OMV definitions by defining query elements for each OMV class, which are connected via sub elements of each referenced OMV class.

The original OMV model is defined via an OWL ontology. For a web service and especially for an ebXML web service it has to be converted to an XML schema. We use the following mechanism to map OWL entities to the appropriate XML schema components for the XML serialisation of the ebXML concepts.

	OWL entity	**ebXML entity**	**XML schema component for WSDL**
Class	Class	Object type	**element with complexType**
attribute	Datatype property (max cardinality = 1)	Slot	**attribute**
attribute	Datatype property (max cardinality > 1)	Slot	**Element with simpleType**
Binary relationship	Object property	Association	**Element with reference type**
Basic data type	XML schema simple data types	XML schema simple data types	**XML schema simple data types**

In principle it is possible to (semi-) automatically generate this conversion. However as it is only needed for the OMV meta ontology it has been done manually up to now.

2.2 NeOn Registry Service Interface

The service is separated into the query interface and the interface for the life cycle operations on the registry objects. Thus the service description (WSDL) contains two port-types *QueryManagerPortType* and *LifeCycleManagerPortType* in the namespace *urn:neon-toolkit-org:registry:omv:service:2.3*. They are specified in the WSDL NeOnRegistryOMV.wsdl. This is an extension of the standard ebXML registry service WSDL. The current specification is based on OMV version 2.3. The only extensions are two additional imports of the following schemas:

- OMV definitions for ontology registry classes.

  They are contained in the XML schema document *rim-omv.xsd* with the target namespace *urn:neon-toolkit-org:registry:omv:xsd:rim:2.3* and the default prefix *rim_omv*.

  In this specific schema subtypes of the ebXML type *rim:RegistryObjectType* are defined for all OMV classes. This is exactly the same mechanism, which ebXML also uses for the predefined provenance model.

- OMV query definitions for ontology registry classes

  They are contained in the XML schema document *query-omv.xsd* with the target namespace *urn:neon-toolkit-org:registry:omv:xsd:query:2.3* and the default prefix *query_omv*.

  In this schema subtypes of the ebXML type *query:RegistryObjectQueryType* are defined. There is a subtype for every OMV class (e.g. *query-omv:OntologyQueryType)*. This allows to directly express the query with XML elements of the OMV specific class instead of having to formulate the query in a general purpose query language with XML syntax. For every datatype property there is an appropriately specialized ebXML filter query, which contains string and numeric comparison operators. For every object property a reference to the query subtype of the range is available. This means an existence test on the object property with its eventually specified filters.

  For example the following query fetches all ontologies, where the number of classes is lower than 500 and for which an OntologyLanguage exists, which has the acronym value *“OWL/DL”:*

```
<query_omv:OntologyQuery>
    <query_omv:NumberOfClassesFilter comparator="LT" value="500”/>
    <query_omv:HasOntologyLanguageQuery>
        <query_omv:AcronymFilter comparator="EQ" value="OWL/DL”/>
    </query_omv:HasOntologyLanguageQuery>
</query_omv:OntologyQuery>
```

2.2.1 Document structure

The following picture illustrates the document structure of the NeOn Ontology registry service.

It extends the rim and the query model schema documents by OMV specific versions. The WSDL for the service imports these two additional schema documents. However the existing ebXML schema documents are still directly or indirectly imported.

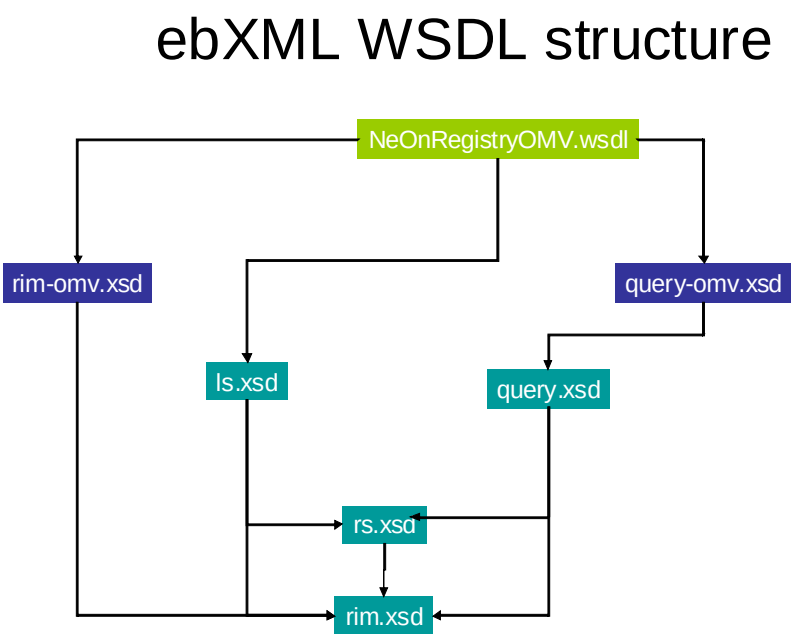


Figure 2.2.1. Document structure of WSDL

2.2.2 Operations

In the following we give a short overview on the different operations (see figure 2.2.2.) of the two interfaces (called porttypes in WSDL 1.0) *LifeCycleManagerPortType* and *QueryManagerPortType* of the registry service.

Figure 2.2.2. Operations

2.2.2.1 SubmitAdhocQuery

The result parameter is a list of the OMV subtypes (rim_omv) of *rim:RegistryObjects*. The input parameter is a query formulation specified by the OMV subtypes (query_omv) of *query:RegistryObjectQuery* like OntologyQuery.

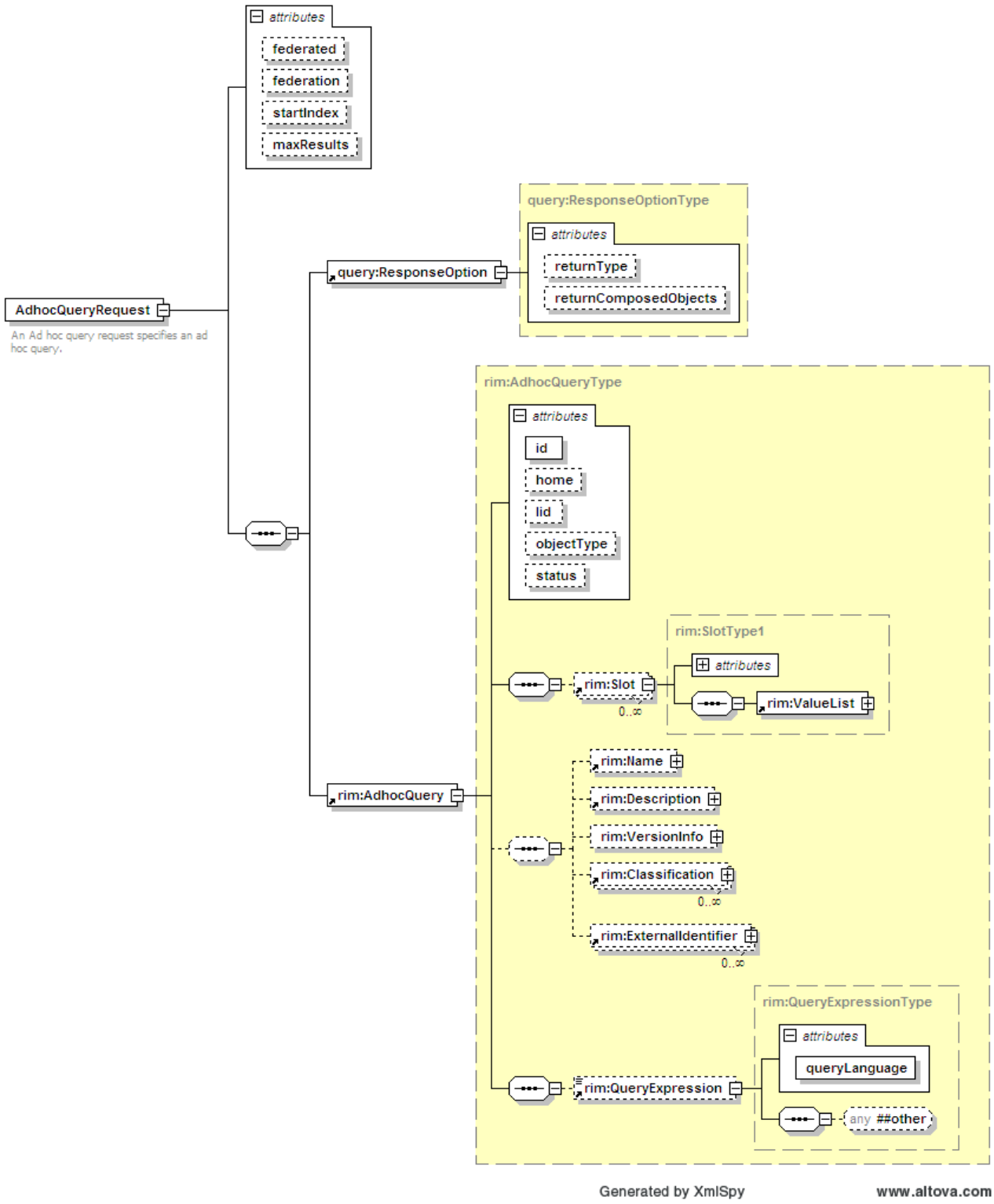


Figure 2.2.3. Query Parameter

The following diagram visualizes some parts of the query elements, which can be specified in the ontology registry query.

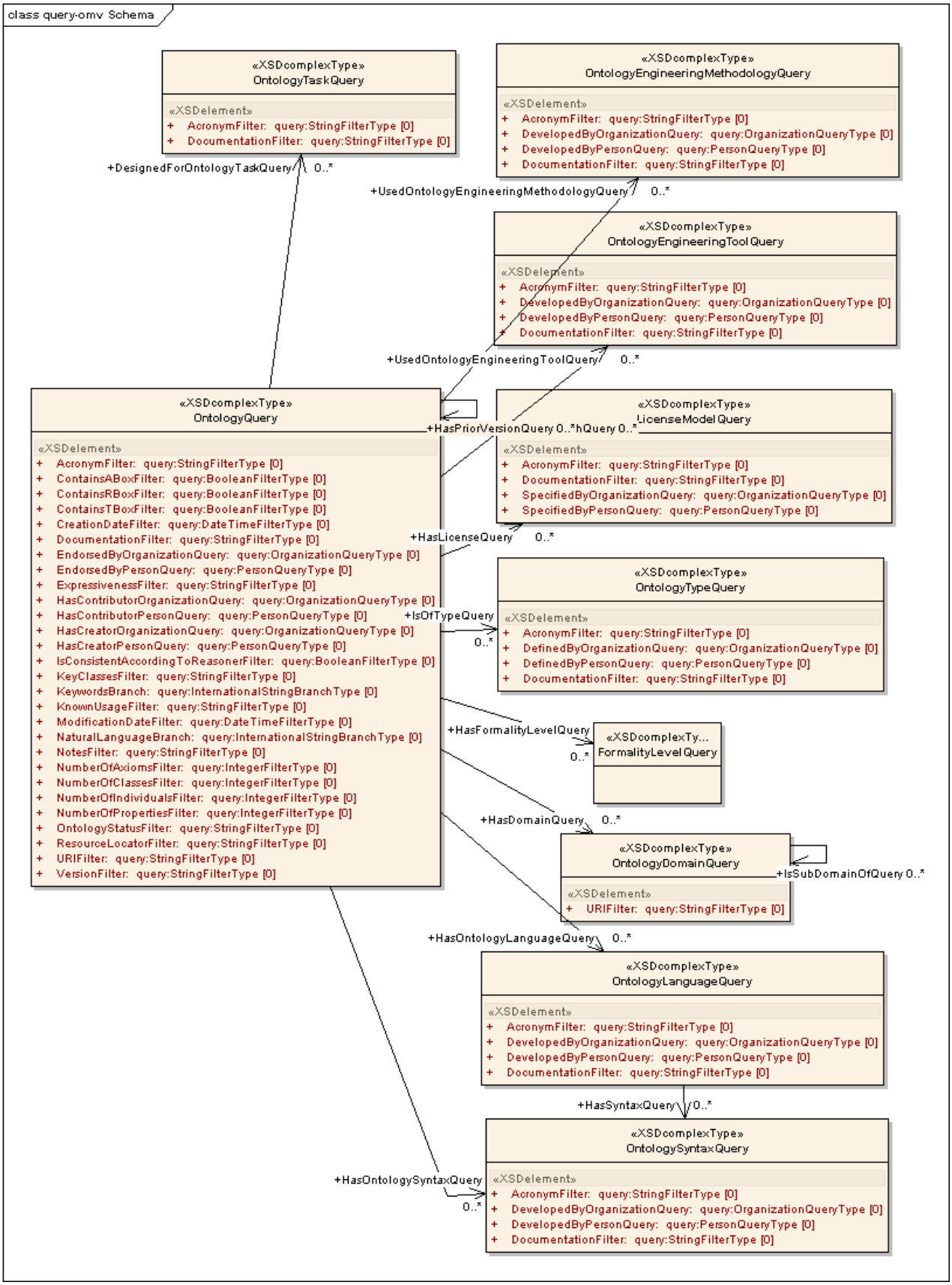


Figure 2.2.4. Query Formulation

2.2.2.2 Example Queries

In the following some SOAP requests for example queries are shown. The essential query formulation is highlighted by bold letters.

Get all registry objects of type ontology

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
xmlns:SOAP-ENC=http://schemas.xmlsoap.org/soap/encoding/
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd=http://www.w3.org/2001/XMLSchema
xmlns:rs="urn:oasis:names:tc:ebxml-regrep:xsd:rs:3.0"
xmlns:rim="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0" xmlns:query="urn:oasis:names:tc:ebxml-regrep:xsd:query:3.0"
xmlns:rim_omv="urn:neon-toolkit-org:registry:omv:xsd:rim:2.3"
xmlns:query_omv="urn:neon-toolkit-org:registry:omv:xsd:query:2.3">
    <SOAP-ENV:Body>
        <query:AdhocQueryRequest id="http://www.test.com/test1">
            <query:ResponseOption/>
            <rim:AdhocQuery id="http://www.test.com/test2">
                <rim:QueryExpression
                    queryLanguage="urn:oasis:names:tc:ebxml-regrep:QueryLanguage:ebRSFilterQuery">
                    <query_omv:OntologyQuery/>
                </rim:QueryExpression>
            </rim:AdhocQuery>
        </query:AdhocQueryRequest>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Get all registry objects of type ontology, which have a certain number of classes.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd=http://www.w3.org/2001/XMLSchema
xmlns:rs="urn:oasis:names:tc:ebxml-regrep:xsd:rs:3.0"
xmlns:rim="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0" xmlns:query="urn:oasis:names:tc:ebxml-regrep:xsd:query:3.0"
xmlns:rim_omv="urn:neon-toolkit-org:registry:omv:xsd:rim:2.3"
xmlns:query_omv="urn:neon-toolkit-org:registry:omv:xsd:query:2.3">
    <SOAP-ENV:Body>
        <query:AdhocQueryRequest id="http://www.test.com/test1">
            <query:ResponseOption/>
            <rim:AdhocQuery id="http://www.test.com/test2">
                <rim:QueryExpression
                    queryLanguage="urn:oasis:names:tc:ebxml-regrep:QueryLanguage:ebRSFilterQuery">
                    <query_omv:OntologyQuery>
                        <query_omv:NumberOfClassesFilter comparator="LT" value="500/>
                    </query_omv:OntologyQuery>
                </rim:QueryExpression>
            </rim:AdhocQuery>
        </query:AdhocQueryRequest>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2.2.2.3 SubmitObjects and Update

The lifecycle operations submitObjects and update have the same interface as they have as input a list of registry objects and return back an indication whether the operation was successful. SubmitObjects will create the specified registry object in the request in the repository. Update will update existing objects, by comparing the object identity given as an urn.

For the NeOn registry service this will be specialized XML serializations of the OMV objects according to the rim_omv schema. The following example shows that this serialization represents a natural modelling of this object types.

Example Ontology registry object

```
<rim-omv:Ontology
        id="uddi:a1c5c620-70ef-11dc-a376-e6b7d52b6ab7"
        acronym="NewOntology"
        hasOntologySyntax="uddi:1cf5ba00-4a74-11dc-aabc-861235dc2a75"
        hasLicense="uddi:d6635200-4a73-11dc-b5e0-9a2b42eec7bc"
        hasFormalityLevel="uddi:5b94fca0-701f-11dc-9a7c-becb85d68ff2"
        numberOfClasses="501"
        numberOfProperties="1000"
        numberOfIndividuals="300"
        numberOfAxioms="30"
        creationDate="2007-09-30T22:00:00.000Z"
        modificationDate="2007-10-30T22:00:00.000Z"
        resourceLocator=http://a.new.ontology.com/ontology.owl
        URI=http://a.new.ontology.com/
        version="Beta version"
        isConsistentAccordingToReasoner="true"
        containsABox="true" containsRBox="false" containsTBox="false"
        notes="This is just an example OMV ontology and does not describe a real ontology."
        xmlns:rim-omv="urn:neon-toolkit-org:registry:omv:xsd:rim:2.3"
        xmlns:rim="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0">
       <rim:Name>
            <rim:LocalizedString xml:lang="en" charset="UTF-8" value="http://a.new.ontology.com/"/>
        </rim:Name>
        <rim:Description>
            <rim:LocalizedString xml:lang="en" charset="UTF-8" value="An example ontology."/>
            <rim:LocalizedString xml:lang="de" charset="UTF-8" value="Eine beispiel Ontologie."/>
          </rim:Description>
          <rim-omv:hasContributor id="uddi:207ff1cc-25c5-544c-415c-5d98ea91060c"/>
          <rim-omv:hasContributor id="uddi:66457a9a-e9df-98a7-0f91-b640b2356cdf"/>
          <rim-omv:hasCreator id="uddi:b9d23910-b426-11da-94dd-a86f2582ed05"/>
          <rim-omv:designedForOntologyTask id="uddi:9574e500-4a74-11dc-803b-81082268edab"/>
          <rim-omv:endorsedBy id="uddi:b9d23910-b426-11da-94dd-a86f2582ed05"/>
</rim-omv:Ontology>
```

2.2.2.4 RemoveObjects

The remove operation removes a set of registry objects. The objects are specified by a list of object references.

2.2.2.5 More Life cycle operations

ebXML has some more life cycle operations, which sets and un-sets the following life cycle states:

approved, deprecate. They will be realised after they have been integrated with the planned NeOn life cycle support.

2.2.3 Common Realisation

The realisation of the NeOn Registry service interface will usually consists of common parts, which deals with the handling of the interface parameters and its mapping to Java objects. This part has been separately realized so that it can be shared by several implementations.

2.2.3.1 Axis 2 skeleton

The Skeleton that is generated from the registry service WSDL and schema is divided into four parts:

- **Service interface and class** – This is the class that holds the actual implementation of the service. For each action there is a method that accepts the incoming message as an argument and returns a response.
- **Service message receivers** – Two receivers are generated – for the query manager and for the lifecycle manager ports. Their task is to validate the incoming and outgoing messages and parse them to and from java classes. They are also responsible to calling the right method for each operation and passing the arguments.
- **Schema classes** – There are two groups of schema classes – those that are typical for ebXML (lcm, query, rim, rs) and those that are specific for OMV (query_omv.xsd, rim_omv.xsd). Each java class corresponds to a xml schema class or type and has methods for parsing information, changing it in java and serializing it again.
- **Resources** – Each service implementation that is done in Axis2 has a service description - services.xml. It is used by the Axis2 environment to invoke the service. The resources also contain the WSDL and the XSD schemas. Finally we have all the jars we need to access CentraSite like the jaxr interface and implementation, UDDI and XQJ libraries and so on.

The whole skeleton is generated automatically from the WSDL using the wsdl2java tool of Axis2. Due to some not yet solved problems with multiple ports in the Axis2 code generation some manual adaption of the generated skeletons is needed.

2.3 Oyster Server Realisation

Oyster is a distributed registry that exploits semantic web techniques in order to provide a solution for exchanging and re-using ontologies and related entities. To achieve this goal, Oyster implements the proposal for metadata standard OMV (see section 2.1.1) as the way to describe ontologies and related entities.

The goal of Oyster is a decentralized knowledge sharing environment using Semantic Web technologies that allows developers to easily share ontologies.

The Oyster system (Oyster is freely available under http://www.ontoware.org/projects/oyster2) was designed using a service-oriented approach, and it provides a set of APIs. Accessing the registry functionalities can be done using directly the API within any application, invoking the web service provided or using the included java-based GUI as a client for the distributed registry.

As part of the design, Oyster follows the approach proposed in [Palma2007] to identify an ontology metadata entry: it uses a tripartite identifier consisting of the Ontology URI plus an optional version information plus the location of the particular ontology implementation annotated.

Currently Oyster implements the OMV core and OMV extensions: P-OMV for describing peers in the distributed network and the mapping extension. More OMV extensions will be supported in the future.

2.3.1 Architecture

The high-level design of the architecture of a single Oyster node is shown in figure 2.3.1

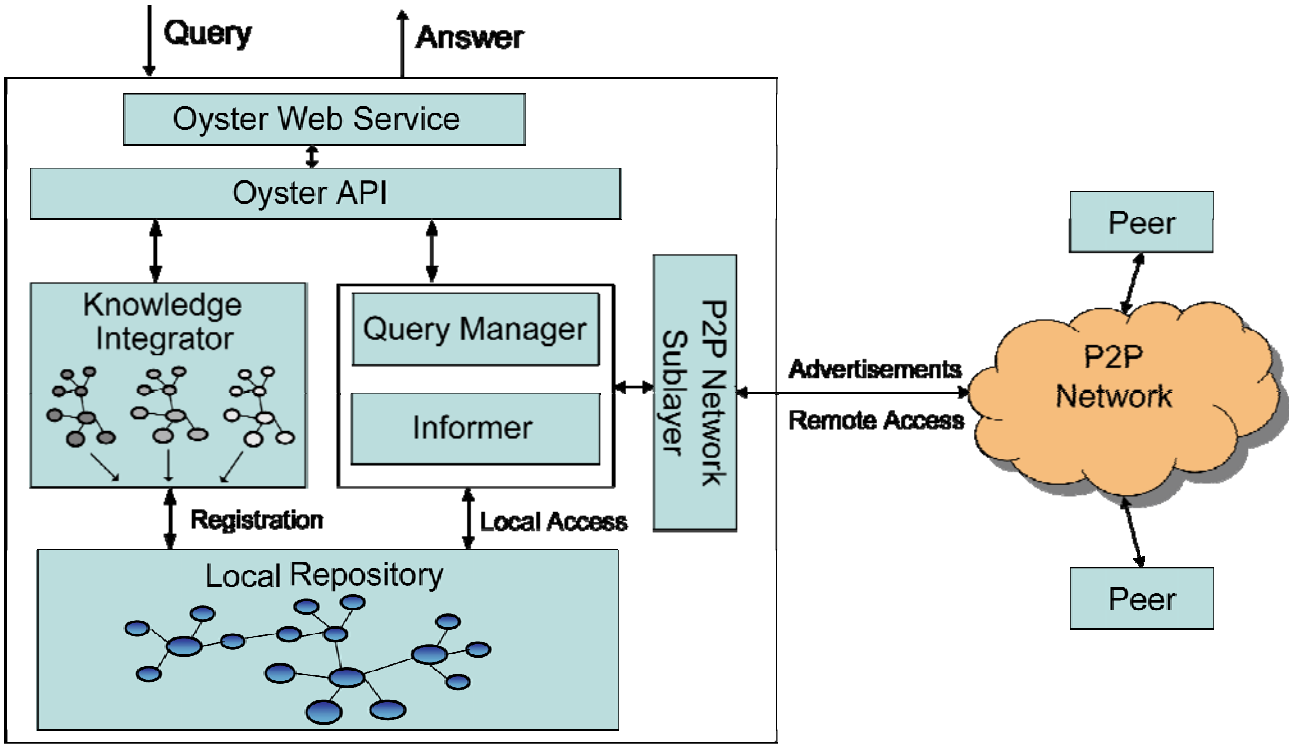


Figure 2.3.1. Oyster Server Architecture

In the following, we briefly discuss the individual components of the system architecture. For a complete description we refer the reader to [Wang2006].

The *Local Repository* of a node contains the metadata about ontologies and related entities (i.e. OMV instances) that it provides to the network. It supports query formulation and processing and

provides the information for peer selection. In Oyster, the Local Repository is based on KAON2 and it supports SPARQL as its query language.

The *Knowledge Integrator* component is responsible for the extraction and integration of knowledge sources (e.g. ontologies) into the Local Repository. This component is also in charge of how duplicate query results are detected and merged.

The Query Manager is the component responsible for the coordination of the process of distributing queries. It receives queries from the user interface, API or from other peers. Either way it tries to answer the query or distribute it further according to the content of the query.

The *Informer* component is in charge of proactively advertise the available knowledge of a Peer in the distributed network and to discover peers along with their expertise (i.e. the expertise contains a set of topics (i.e. ontology domains) that the peer is an expert in).

The Peer-to-Peer network sub-layer is the component responsible for the network communication between peers. In Oyster, we rely on an RMI based implementation; however, other communication protocols would be possible as well.

The *Oyster API* provides a well defined interface with a set of methods that expose all the registry functionalities (see section 2.4.3).

The *Oyster Web Service* encapsulates the Oyster API and provides a free realisation of the NeOn registry service (see section 2.4.4).

Additional registry functionality can be provided by engineering components. Some of these components are described in [PalmaHaase2007].

Finally, the graphical user interface for the registry is introduced in section 2.6.2.

2.3.2 Implementation

The Oyster Web Service realizes the NeOn registry service, which is an ebXML registry web service specialised for ontologies. Following the ebXML specification, our web service defines the ports (LifeCycleManagerPort and QueryManagerPort.) and operations described in section 2.2. The web service[1] was developed using the axis2 technology and uses the Oyster API for the implementation of its operations. In the following, we briefly describe the Oyster API and how the web service uses it to implement its operations.

2.3.2.1 Oyster API

Since Oyster Web Service relies on the Oyster API for the implementation of the service operations, in the remainder of this section we briefly introduce the Oyster API.

OMV, as the NeOn standard for describing ontologies, is the representation used by NeOn applications for exchanging ontology metadata. Therefore, the classes defined by the OMV Ontology have also been converted into java objects to provide a standard OMV API[2] that can be used by java developers for the management of OMV objects (e.g. get/set properties). Figure 2.3.2 illustrates how the API looks like.

[1] Oyster web service is freely available at http://www.ontoware.org/projects/oyster2

[2] The OMV API is freely available at http://www.ontoware.org/projects/oyster2

Figure 2.3.2. OMV classes

The oyster API[3] exposes all the functionalities of the registry. It provides among others methods for the connection management, register metadata, update metadata, remove metadata and query metadata. Oyster API uses the OMV objects defined in the OMV API. In the following we provide an overview of the methods provided:

Connection management

There are three methods for creating a new connection to Oyster that allows the user to customize the way to connect to Oyster (e.g. the location of the preference file, specify to start an instance of KAON2 server internally or not, etc.). Additionally there is a method for closing the Oyster connection.

Register metadata

There are two methods to register metadata in Oyster: either specifying the complete metadata explicitly or specifying the ontology that should be imported by Oyster to automatically extract the ontology metadata.

Update metadata

Replaces an existing OMV object in Oyster registry with the new information provided. If the object does not exists, it is registered

Remove metadata

Removes an existing OMV object in Oyster registry

[3] Oyster API is freely available at http://www.ontoware.org/projects/oyster2

Search metadata

There are two generic search methods: search Oyster registry to retrieve all available metadata entries that fulfil certain conditions or send a SPARQL query to Oyster registry and return the metadata entries that are part of the result.

As the main objects in oyster registry are the ontology metadata entries, we have also methods for searching ontology specific metadata e.g. to get all ontologies, or get all ontologies registered since a specific date or get all available ontology metadata entries that contains the input keyword in any part of any of their data properties. Similarly Oyster provides methods for searching mapping specific metadata.

Additional methods

The API has also methods to get information about the peers in the distributed network and methods for the management of the OMV objects (e.g. translate an OMV object to an RDF file or serialize an OMV object to a string).

2.3.2.2 QueryManager Implementation

The only operation of the QueryManager is implemented in Oyster realisation as follows:

- The input object (OMV query object defined in the XML schema as an extension of the ebXML registry object query) is translated into a SPARQL query. In the current implementation some information in the query object like the LocalizedString for the description or name attributes is simplified as a simple String and the language (xml:lang) or the charset is ignored. In the future this could be improved.
- The submitAdHocQuery method of the Oyster API is executed with the SPARQL query.
- The result of the query consisting of a set of OMV object (defined in OMV API) is translated to the OMV objects defined in the XML schema as an extension of the ebXML registry object.

2.3.2.3 LifeCycleManager Implementation

For the implementation of the LifeCycleManager operations, Oyster web service uses Oyster API as follows:

submitObjects

- The input object (OMV object defined in the XML schema as an extension of the ebXML registry object) is translated into an OMV Object (defined in OMV API).
- The register method of the Oyster API is executed with the translated object.

updateObjects

- The input object (OMV object defined in the XML schema as an extension of the ebXML registry object) is translated into an OMV Object (defined in OMV API).
- The replace method of the Oyster API is executed with the translated object.

removeObjects

- The input object (OMV object defined in the XML schema as an extension of the ebXML registry object) is translated into an OMV Object (defined in OMV API).
- The remove method of the Oyster API is executed with the translated object.

Figure 2.3.3. shows the package structure of the implementation of the Oyster Web Service. The detailed instructions of the installation process and usage examples of the Oyster Web Service can be found in the Appendix of this deliverable.

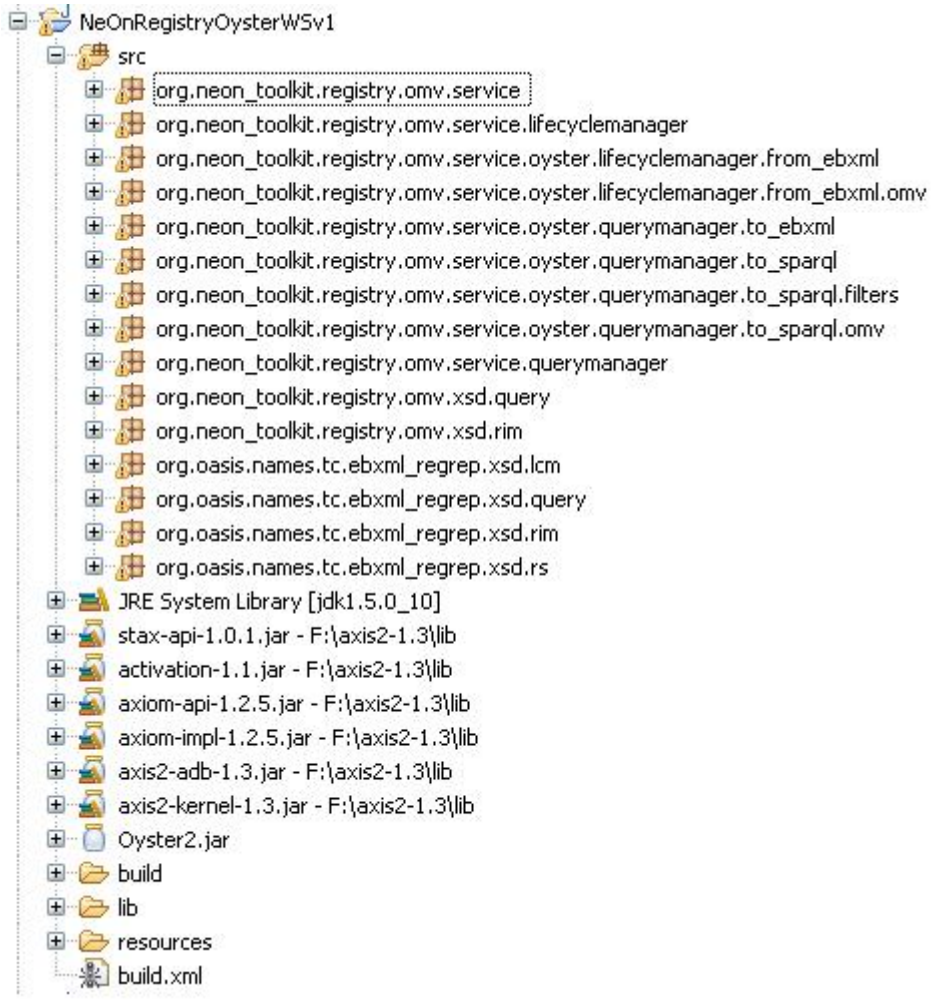


Figure 2.3.3. Package Structure

2.4 CentraSite Server Realisation

The CentraSite based realisation of the NeOn registry service integrates this service into a general service registry repository. As CentraSite is a central part of the larger webMethods SOA suite from Software AG this realisation offers the possibility to combine ontology registry functionality with basically the complete functionality to develop SOA applications.

2.4.1 Architecture

The CentraSite Server realisation uses the JAXR interface to realize the registry web services. Thus, it results in the following architecture.

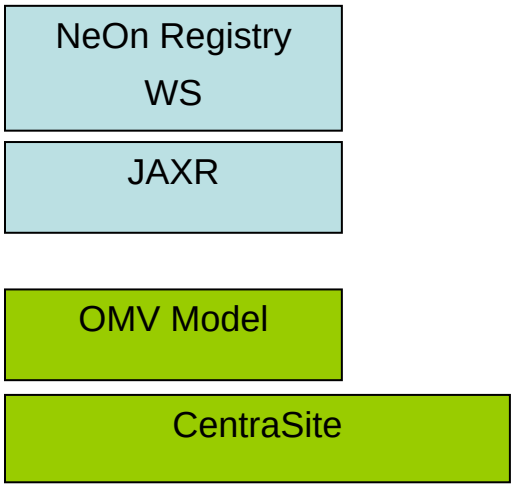


Figure 2.4.1. CentraSite NeOn registry realisation

We can distinguish the following components, which are explained shortly.

2.4.1.1 CentraSite

CentraSite is a general purpose registry and repository. It is especially suited for SOA artefacts like web services and XML schemas. It usually operates as a central server with capabilities for mission critical applications. It has also built-in federation capabilities and is based on open standards.

Where the repository serves as storage area for certain objects, the registry stores the metadata on these objects. An additional area of application for CentraSite is that of registering (web-) services in particular, and other SOA artefacts in general. For semantic applications this means that the CentraSite registry shall be the single and authoritative place where all the relevant information on ontologies and other artefacts including web services can be found.

If the metadata modelling is done properly, the artefacts will contain references to other entities, thus documenting which is using or is being used by the other. Given this information, CentraSite can generate graphical impact analysis reports that show in detail who and what will be affected by a change on a particular object. Additionally, individual users can subscribe to be notified about changes on arbitrary objects within the registry. As part of a lifecycle management infrastructure, a ‘change’ in the above sense might even be the administrative decision to mark a certain service as ‘deprecated’. This will enable a developer to take the proper steps if a service they rely upon is

bound to change. Depending on the use case, CentraSite is accessible through a number of relevant interfaces. “End users” can access both the registry and the repository through a AJAX base browser GUI component. The same functionality is also available as Eclipse plugins, which allows access to the areas of CentraSite that are relevant to this role – relieving them of both having to change tools to obtain relevant information and of having to wade through screens of data that is not applicable or important at the time. For automated access, CentraSite is accessible via a UDDI and a JAXR interface.

Features that are common to both the registry and the repository aspect of CentraSite are:

- Auditing

  Any persistent change to any kind of an artefact is logged inside CentraSite. Logged information currently consists of the user and the time at which the action took place.

- Versioning

  Unless configured differently, a complete version history of all artefacts is maintained inside CentraSite.

- User Role Management

  CentraSite users are governed by a full fledged role based user management system. It allows a finely adjustable definition of roles and their rights in terms of object manipulation (create, read, update, delete). An example of the granularity and the possibilities can be seen in the Life Cycle Management plugin of CentraSite. The general scenario here is to control which type of user (Architects, Developers, and Deployers) can define which type of manipulation on a particular service object. For example, a developer can change state of service from “designed” to “implemented”, but not to “retired”, whereas an architect would not be allowed to change the state to “implemented” – simply because it is assumed that persons in the role of an architect will not do the implementation themselves. While, as stated above, the authorization is easily controllable and adjustable, authentication is done via a number of standard mechanisms, such as LDAP or Active Directory Server (ADS).

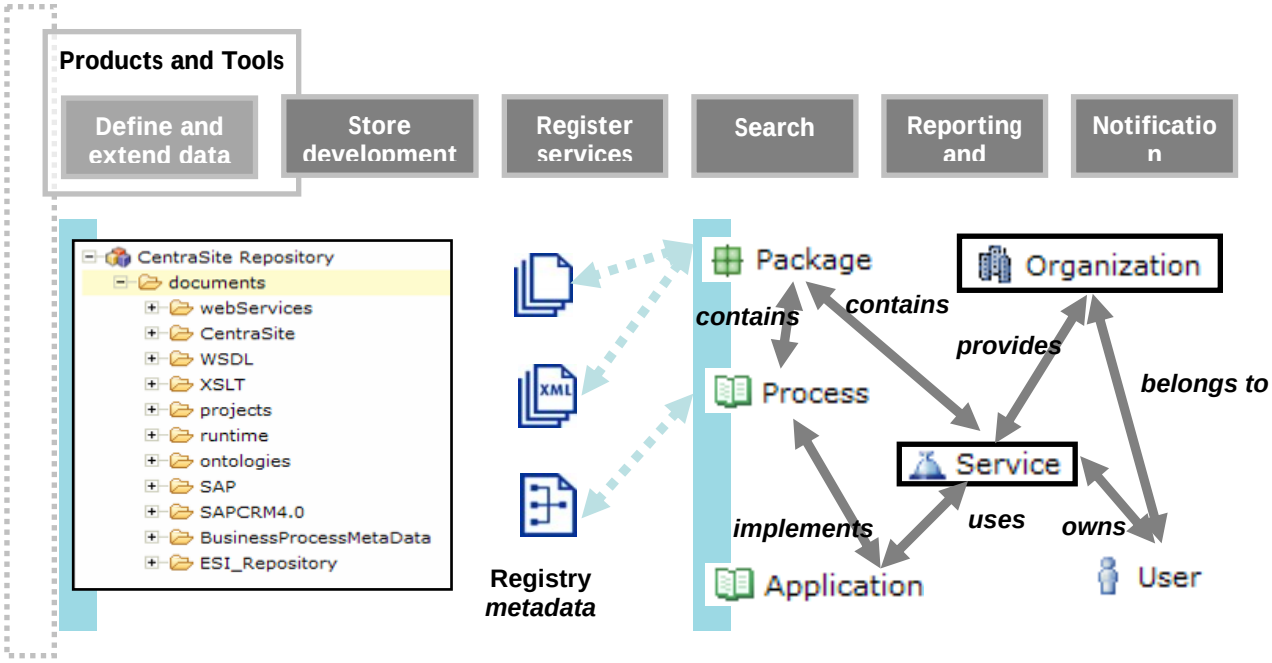


Figure 2.4.2. CentraSite General Architecture

2.4.1.2 CentraSite with OMV Model

CentraSite has been extended with the OMV model. This results in a generic JAXR Java interface to the OMV model. This requires to use always general purpose mechanisms to access the OMV model. However, it has the advantage that the OMV model is completely integrated with any other registry model like web services. This allows analysis of the relationship general purpose objects like web services and databases together with ontologies.

2.4.1.3 JAXR

The realisation of the general purpose JAXR interface is available in an application server

2.4.1.4 NeOn Registry Web Service CentraSite

Based on this general purpose JAXR interface the NeOn registry web service is realized as a web application. Thus it can be deployed into the same web application server as the underlying JAXR implementation.

Compared to the JAXR interface this offers the registry functionality now both as web service and as specialized NeOn registry service. Thus the service directly processes NeOn registry objects in XML serialisation.

2.4.2 Implementation

In the implementation of the service we have used Axis2 to generate a web service skeleton from the WSDL and the schemas. The service functionality itself is realized though a series of classes that convert the ebXML objects and queries to JAXR and the vice-versa.

2.4.2.1 QueryManager Implementation

The only operation of the QueryManager is *submitAdhocQuery.* There are two main tasks in the implementation – generate a query from the service message, which is understandable by CentraSite, and convert the returned JAXR objects in the corresponding ebXML elements.

Due to the nested structure of the ebXML *ebRSFilterQuery* we have build a series of java classes that translate each query element into an XQuery predicate. The filter classes add simple conditions to the predicate, while the more complex classes could add subqueries that match the nested ebXML queries. When generating the predicate, only the filters and subqueries that are specific to OMV are supported. Other ebXML filters are ignored, such as *ClassificationQuery*, *AssociationQuery* and *SlotBranch*.

For the execution of the query a CentraSite extension of the JAXR interface is used - CentraSiteQueryManager. It allows querying the CentraSite registry with custom XQueries and returns JAXR java classes that correspond to the resulting registry objects.

While it is possible to map the result of the XQuery direct to the ebXML schema, the current implementation does that manually. For the resulting objects, each property is copied one by one. Again only OMV objects are supported. The whole process is done inside a single class: *org.neon_toolkit.registry.omv.service.centrasite.querymanager.to_ebxml.ebXMLTranslator*

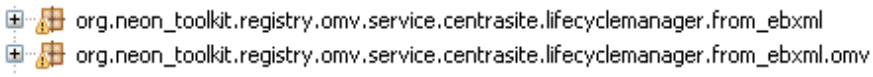
The supported return types are *LeafClass* and *ObjectRef.* In the first case the whole OMV object is returned with the object properties as references. In the second case, only the UDDI key of the object is returned.

2.4.2.2 LifeCycleManager Implementation

The operations of the LifeCycleManager can be divided into two groups:

- submitObjects and updateObjects

The first group requires the full objects to be passed in the incoming message. For each object, the service searches for a matching key in the registry. If one is found, it is updated with the new properties (in both operations). If there is no such object in the registry and the operation is *submitObjects,* then a new object is created. Otherwise the update is ignored.



The implementation of these operations is done in two packages. The classes in the first one are responsible for mapping standard ebXML types like InternationalString, Classification and Association to the JAXR types. The second one maps the OMV objects.

Note that the object properties are represented in ebXML as object references. In this implementation they are translated into associations. When list of references for an object property is found in the request, all former associations for this property are deleted and the new are created in their place. If no references are specified in the request for an object property, the old associations are kept.

Similar to the query manager, all ebXML specific properties like classification, association and slots are ignored. The OMV data and object properties, as specified in the OMV ebXML schema, are mapped to slots and associations in CentraSite.

If no exception is returned by CentraSite or the Axis2 skeleton, the service returns a message with the status *Success*.

- approveObjects, removeObjects, deprecateObjects and undeprecateObjects

The second group requires only a list of keys as an input. The JAXR API has most of the methods in its *BusinessLifeCycleManager.* The *CentraSiteLifeCycleManager* add the method *approveObjects*. Thus the implementation of these operations simply passes the key list to the corresponding methods in JAXR. If some of the keys do not match an existing object in the registry, they are ignores.

Again, if no exception is returned by CentraSite or the Axis2 skeleton, the service returns a message with the status *Success*.

2.5 Positioning Oyster server and CentraSite server

With the Oyster and CentraSite server two realisation of the NeOn registry service are available. Each of them has a quite different focus and is suitable for different applications. In addition their interoperability with common clients shows the suitability of the defined NeOn registry service interface.

Oyster registry server

The Oyster server is a pure NeOn ontology registry server. The server offers just this functionality.

It is an open source realisation, which is freely distributed. It relies on a more research-oriented P2P infrastructure.

CentraSite registry server

The CentraSite registry server is a general purpose registry and repository. Thus, it offers besides the ontology registry a lot of other functionality.

It is a commercial realisation. Thus, it has capabilities for mission-critical applications. However, there is also a freely available community edition of CentraSite. This is however not open source and has some functional restrictions.

As a default CentraSite is used as a central server. There are however additional federation capabilities.

2.6 Outlook: usage of the NeOn Ontology registry service

The web service form of the NeOn registry service is of course one of the most flexible interfaces, which can be used in all kinds of environments. We want to point to two planned usages of the web service in the NeOn project.

2.6.1 Java interface

In order to use the web service functionality in a Java program an API is very useful. It is planned to define and implement such an API in a future version of the NeOn toolkit infrastructure deliverables. It will be along the lines of the Oyster API and also a specialization of the general purpose Java registry API JAXR.

2.6.2 NeOn Registry Plugins

A very important use of the NeOn registry service will be of course in NeOn engineering and GUI components. In the following we briefly introduce two plugins for the NeOn registry that allows different levels of access to the registry functionalities while providing a bigger flexibility to the users: The first is a complete GUI component that provides access to all the registry functionalities and is under development. The second plugin is an import wizard that provides basic access to the registry information without having to change the user perspective in the NeOn toolkit and is already available at http://www.ontoware.org/projects/oyster2.

2.6.2.1 Oyster based Ontology Registry GUI component

The *Oyster EC (Engineering Components)* provides additional or specialized functionality over the registry (e.g. change management). A description of some of these components can be found in [PalmaHaase2007].

The Oyster GUIs are a set of graphical interfaces for the registry and engineering components. For example, the main registry GUI provides a ready to use client interface to the distributed registry. It implements most of the functionalities provided by the Web Service and offers to the final user a way to interact with the registry.

The NeOn registry GUI will be implemented as a NeOn toolkit plugin (i.e. a perspective extension) based on the current Oyster GUI. It will provide a common client to the two NeOn registry implementations: Oyster and CentraSite. For the implementation, the GUI will use the common Web Service interface described in section 2.2. Figure 2.6.1 illustrates the look and feel of the graphical interface. In the following we provide a brief description of its envisioned functionalities:

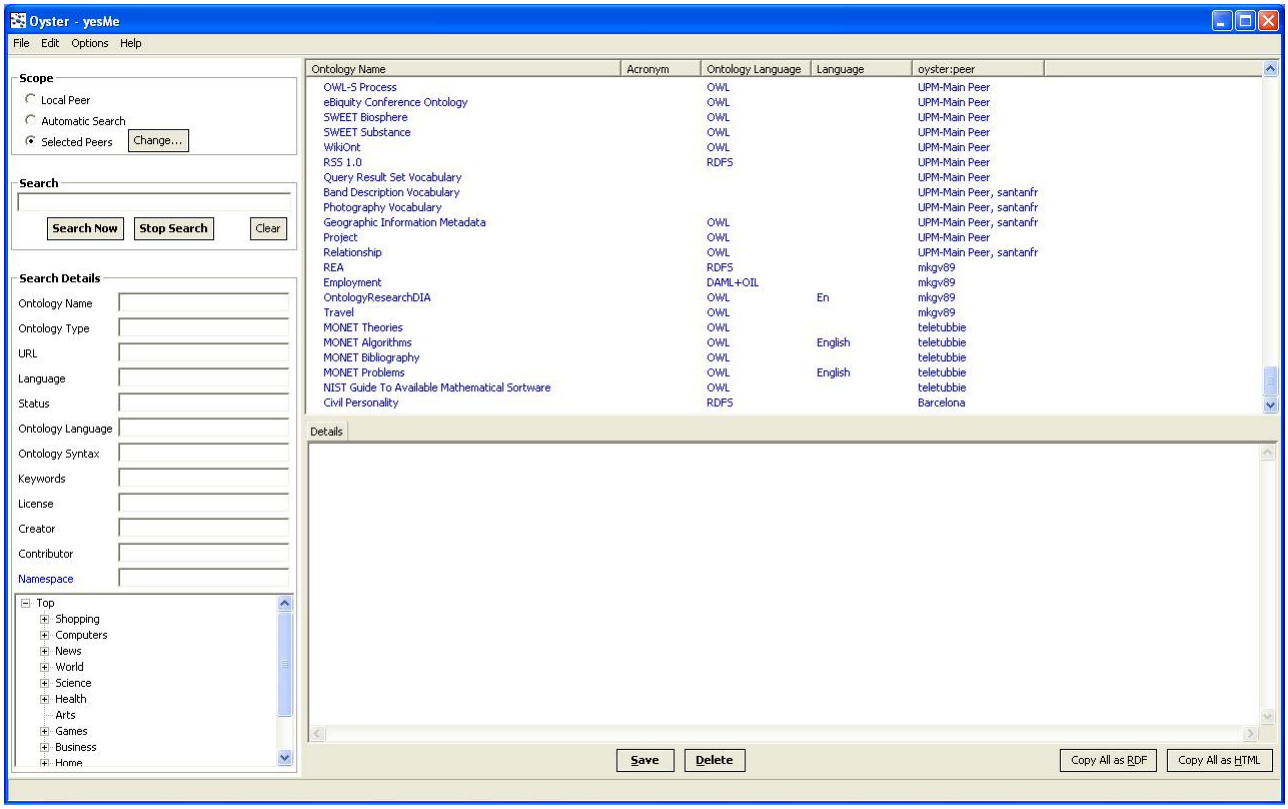


Figure 2.6.1. GUI of NeOn toolkit plugin

1. **Publishing and Importing Metadata** The GUI enables users to register metadata about ontologies manually and also to import ontology files to extract the ontology metadata available and let the users specify the missing values.
2. **Formulating Queries** As shown in the left pane of the figure 2.6.1, users can use the GUI to search the registry for ontology metadata by means of simple keyword searches, or more advanced semantic queries like SPARQL queries. Queries are formulated in terms of two ontologies (1) the proposal for a metadata standard OMV that describes the properties of the ontology, and (2) a topic hierarchy (i.e. DMOZ) that describes specific categories of subjects to define the domain of the ontology.. The GUI includes by default some of the most common properties used for searching ontologies like name, acronym, ontology language, etc. Additionally, it also includes a way to search ontologies by topic terms.

3. **Routing Queries** In the case of a distributed registry (i.e. Oyster), as shown in the upper left pane of the figure 2.6.1, the GUI provides the means for the users to specify if they want to query a single specific peer, a specific set of peers, or the entire network of peers.
4. **Processing results** The results matching a query are presented in a result list (c.f. upper right pane in figure 2.6.1). In a distributed registry like Oyster, the answer of a query might be very large and may contain many duplicates due to the distributed nature and potentially large size of the Peer to Peer network. For a description of how the Oyster implementation handles such duplicates, please refer to [Wang2006]. The details of particular results are shown in the lower right of the figure 2.6.1. Additionally users can save the results of a query into their local repository for future use.

2.6.2.2 A basic NeOn toolkit registry plugin

For the first prototype of the NeOn toolkit, we developed an import wizard plugin for the registry that allows the users to search for ontologies based on their metadata. The wizard returns to the user a list of ontologies (i.e. ontology metadata) that fulfil the conditions and allows the user to select and import an ontology into the workspace. The current version of the import wizard is based on the Oyster realization of the NeOn registry and uses the Oyster API. The import wizard plugin is freely available at http://www.ontoware.org/projects/oyster2. The following figures show how the import wizard works.

1. Launch the import wizard

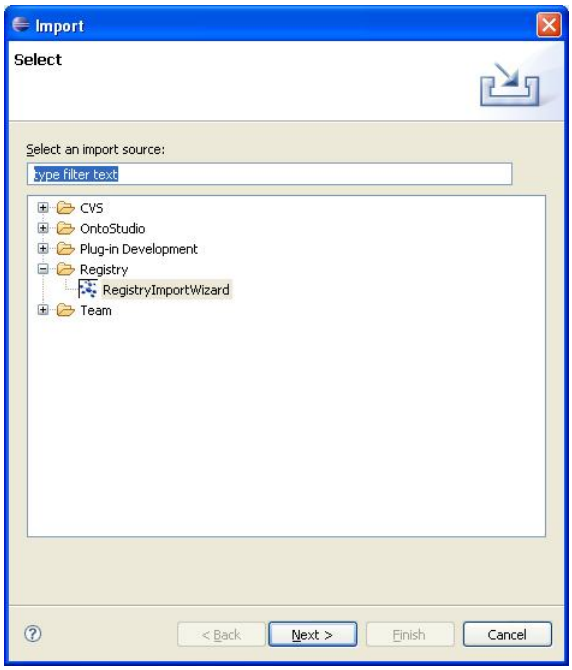


Figure 2.6.2. Launch of Import Wizard

2. Specify the conditions (i.e. OMV properties) that should be fulfilled by the ontologies we are searching.

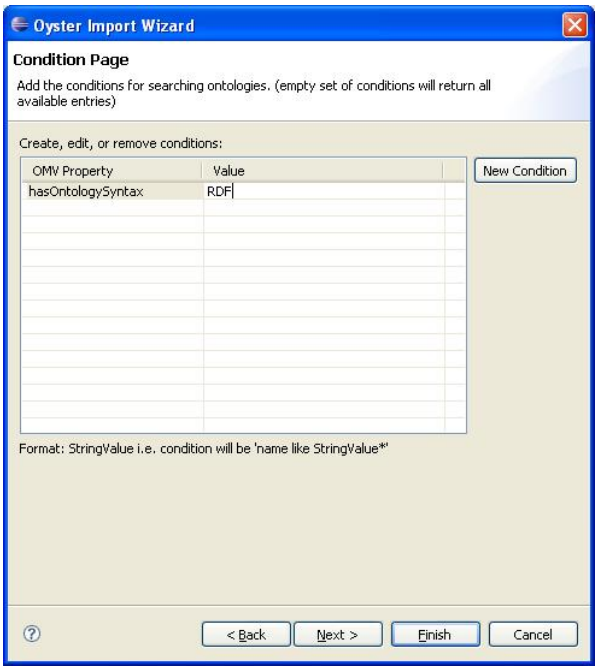


Figure 2.6.3. Query Formulation

3. Select an ontology from the result list

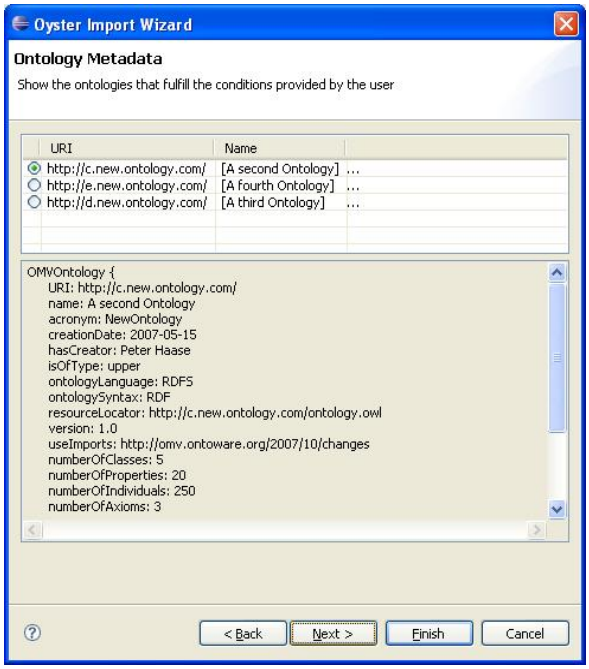


Figure 2.6.4. Ontology Selection

4. Select the NeOn project in which will be imported the selected ontology. Note that the selected entry is the OMV metadata (not the ontology itself). However to import the ontology into the workspace we use the resourceLocator information that is the location where the actual ontology can be found.

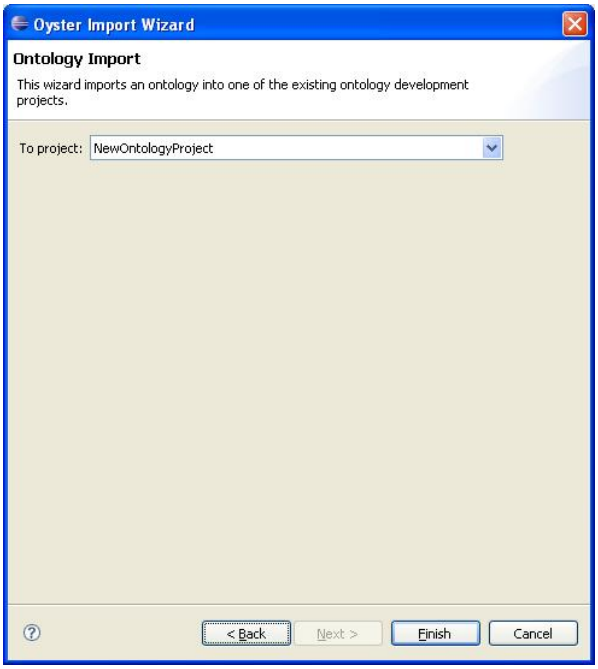


Figure 2.6.5. Import Ontology in development environment

3 Ontology Repository

The current NeOn ontology repository is focused on ontology development support. Thus it follows a documented oriented approach by treating the ontology as one document. The integration of a more runtime-oriented approach to be integrated with reasoners is planned for a future version by defining a common (Java) interface based on the NeOn data model for runtime- and development-oriented repositories. Thus we cover here only the basic repository functionality and its integration with the registry.

Basic repository functionality

The repository manages ontologies identified by a unique name given as an URI in a persistent store. They are organized in a hierarchical, directory-like structure of collections. The ontology repository manages directly all the artefacts needed for an ontology-based application. This consists of the following kind of data:

- Ontologies according to the NeOn ontology model. This includes the OWL and the rule model but also the mapping and module model extensions
- Other data in the following formats, where the differentiation is based on the mime-type.
    - XML
    - text
    - Binary

Basic Retrieval

The basic functionality is to access complete ontologies in a persistent store. It requires to identify the repository container itself. This is done by an URL. There are two basic mechanisms to access the ontology:

- Direct access
- Navigation

The direct access allows to directly access the ontology by an URL, which specifies its location in the repository. Thus it is usually different to its unique name URI.

With the navigation it is possible to traverse the hierarchical structure of collections, where the ontologies are located. This includes navigation to parent, children, successor and predecessor.

Manipulation operations

The basic manipulation operations are:

- Creating ontology in a specified collection
- Updating ontology given by location
- Deleting ontology given by location

The repository has the following service interfaces:

- WebDAV
- Integrated repository and registry web services (see below)

CentraSite Repository Functionality

The primary intention of CentraSite Repository functionality is to store all kinds of development artefacts such as ontologies. Above and beyond the aspects of pure storage and retrieval, CentraSite offers mechanisms that digitally sign and timestamp to guarantee authenticity and no repudiation. In CentraSite, access is done through the WebDAV interface. Technically speaking, it is an extension to the HTTP protocol, and enables collaborative access to arbitrary documents on a remote storage.

Integration of repository and registry functionality

For several applications an integration of the repository and registry functionality means a much simpler handling compared to the invocation of the two interfaces. Therefore we offer an integrated service interface, which consumes and delivers the OMV registry objects together with the ontologies themselves.

At the interface level this means that we rely on the previously described registry web services of the lifecycle manager and query manager porttype. To these operations a list of ontologies will be added. In the web service this is as an attachment as an additional optional parameter.

Automatic derivation of registry objects

An extension of this mechanism is to automatically derive the computable OMV registry information from the ontology and then also store both information in either the registry or the repository. This is handled by providing a content cataloguing service, which handles the automatic extraction of ontologies in ontology metadata (see Figure 3.1.1.).

Figure 3.1.1. Content Cataloguing Service

Thus this is the most comfortable service, where only the ontology is provided and registry objects are created automatically. This is however sufficient only for the very basic registry information. A large part of the OMV information cannot be automatically computed and thus has to be provided via the conventional lifecycle manager operations.

4 Qualitative Evaluation

In this section we provide a preliminary evaluation of the NeOn registry and repository. As the developed software has not yet been applied in the NeOn case studies yet, we do not provide a performance evaluation yet.

Instead we evaluate the software against the requirements for the NeOn architecture as they have been defined in deliverable D6.1.1 [Gomez2006]. We restrict the discussion to the requirements related to the registry and repository and discuss to what extent the requirements are fulfilled. If a requirement is not fulfilled we explain the reasons and potential future steps needed to be taken.

It should be noted that the terminology in the requirements document slightly differs from that the one used in this document. For example, in the requirements document no clear distinction between registry and repository was made, instead the term repository was used in a more general sense.System Requirements

R2.1.1.1: Information resources will be potentially stored in a distributed repository managed by this architecture layer. NeOn shall access these resources transparently to their location.

This is the main functionality provided by the registry and repository. Both the Oyster and CentraSite implementations are distributed systems. CentraSite allows for federation, while Oyster inherently is inherently a decentralized network of individual Oyster nodes.

R2.1.1.2: The middleware layer between the NeOn toolkit and the distributed repository shall implement a number of services which allow users to exploit the information stored in the distributed repository.

As described, the registry provides mainly two sets of services: one for the lifecycle management of registry objects, one for querying the content of the registry.

R2.1.1.5: Support for integration of heterogeneous information sources

The following types of distributed information resources shall be integrated in the NeOn knowledge model:

- Unstructured content, e.g. text.

- Structured knowledge with no clear semantics, e.g. DBs, catalogues.

- Formal knowledge (ontologies and data)

- Semantic annotations

In the current version, the registry focuses on ontologies as main artefacts to be managed. In principle – being based on ebXML – it can be used for arbitrary other artefacts. If needed, in the future we may extend the registry with additional specializations.

R2.1.1.13 Provenance service

The NeOn middleware layer shall include a service which keeps track of the provenance of ontologies, i.e. how, when, and by whom ontologies evolve across their lifecycle.

The lifecycle and provenance of the ontologies are explicitly captured as metadata in OMV and thus managed in the registry.

R2.1.1.16 Dual language approach

Description logic-based languages, e.g. OWL, and Frame/Rule languages, e.g. FLogic, shall be supported

With the metadata schema of OMV, ontologies in arbitrary ontology languages can be described. The ontology language is captured as a special metadata attribute. As such, the dual language approach is supported.

4.1 User Interface Requirements

The user interface requirements state that NeOn shall provide user interfaces to every NeOn service R2.1.2.3.3.

The implementations of the registry provide a user interface to support the selection and reuse of ontologies, as described in the previous chapters.

One requirement that is not addressed is that of R2.1.2.3.4 Access management to information resources: NeOn shall provide a GUI to show and manage access rights to resources.

While CentraSite itself provides access management, this functionality currently has not been addressed in the implementation of the NeOn architecture at all. It will be left for future versions.

4.2 Communication Interfaces

R2.1.5.1 Service oriented architecture

NeOn shall offer a service oriented interface in order to ease the inclusion of new components into any of the three architecture layers

R2.1.5.2 API –based interface

The NeOn backend will be accessible by means of several client interfaces, e.g. Java, .NET.

The registry provides both a Web Service interface (described in WSDL) and a Java client interface. It thus meets the requirements of a service oriented interface and a client interface for tight coupling. A .NET compatible interface is not envisioned.

4.3 Specific Requirements

R3.1.2 Extension of state-of-the-art Semantic technology

R3.1.2.2 NeOn shall re-use and extend KAON

R3.1.2.4 NeOn shall re-use and extend Ontobroker and Ontostudio

R3.1.2.6 XML-based repository: NeOn shall re-use and extend XMLbased repository from SAG

The relevant existing technologies have been reused in the implementation: The Oyster system relies on the KAON2 API (the standard NeOn API for ontology management, which by is supported and used by both KAON2 and Ontobroker), while the CentraSite implementation is a direct extension of SAG's XML based repository.

4.4 Detailed Functions

R3.2.1 Access to distributed repository: Transparent access to information resources

NeOn shall allow access to ontological and non ontological information pieces transparently.

Distributed Repository access API: NeOn distributed repository API shall provide access to knowledge contained in the repository

Resource transparent storage: NeOn shall provide transparent access to resource storage. The NeOn Distributed Repository shall internally manage the physical location where the resource is actually stored.

The APIs defined for the registry and repository provide a completely transparent access to the ontologies. OMV provides the means for logical identification of ontologies, independent of their physical location. The complete details about logical identification and physical location are described in the OMV specification.

R3.2.1.4.1 Access to distributed repository

High availability of information resources

The NeOn architecture shall automatically distribute replicas of information resources to facilitate local access to information.

NeOn shall implement a device to ensure consistency of distributed replicas

Currently, we do not provide means for automatic replication. However, the decentralized implementation allows to (manually) manage information about replica: An ontology with a unique logical identifier can be held in different physical locations. Using the change propagation techniques described in D1.3.1 [PalmaHaase2007], the ontologies and their metadata can be kept up-to-date.

R3.2.11.1 Lifecycle support for ontology development

Extended ontology support to the networked paradigm

NeOn shall extend standard ontology development facilities to the distributed, networked case

NeOn shall provide the means to access one or several ontology repositories for ontology selection. Then, the selected ontology/ies will be imported by means of the ontology load functionality

NeOn shall allow to select and reuse ontologies and ontology entities from a network of ontologies

The extensions to support networked ontologies are built directly into OMV: OMV provides the required metadata attributes to describe the relations of an ontology within a network of ontologies. These metadata can be queried using the APIs provided.

4.4.1 Performance

So far, performance evaluations have not been performed yet. The performance evaluations will be performed in practical settings, once the software is applied in case studies. We here only outline the relevant criteria that will be used for the performance evaluations:

Scalability (R3.3.1): Scalability will be important in both the dimension of number of distributed registries (nodes) and the number of ontologies stored at each registry.

For the performance measurements, the times for (1) loading and storing, and (2) querying of ontological resources will be relevant.

5 Appendix

5.1 Appendix 1: Installation Oyster Registry

1. Download the compressed file NeOnRegistryOMVOyster_vx.y.zip from http://www.ontoware.org/projects/oyster2 where x.y is the version number of the release. The compressed file contains the following installation files:
    a. NeOnRegistryOMVOyster.aar: Is the Oyster2 Web Service
    b. Oyster2APIv0.98.zip: contains the following required files and directories:
        i. new store: The configuration file
        ii. O2ServerFiles: This folder includes all the necessary ontology files for Oyster: OMV core and extensions and topic hierarchy.
        iii. Oyster2: This folder includes two files: kaon2.jar and run.bat
2. Extract the compressed files into the Oyster2API_vx.y folder. The default location is c:\ (e.g. c:\Oyster2APIv0.98). To customize the location see section 5.1.1.
3. Copy the file "new store" to the working directory of the web server (e.g. jakarta-tomcat-5.0.28\bin).
4. Edit the file “new store” to specify the name of the current node (i.e. property localPeerName).
5. Execute the batch file run.bat to start a KAON2 instance.
6. Copy the aar file into the service container of axis2 (e.g. jakarta-tomcat-5.0.28\webapps\axis2\WEB-INF\services).
7. Start the web server.

5.1.1 Customize the location of Oyster files

To change the default location c:\ (e.g. to another windows folder or to another operating system path), two files have to be edited:

- new store: Edit the path location of the required files.
- run.bat: Edit the path location of the kaon2.jar file and the location of the server container of the KAON2 instance.

5.2 Appendix 2: Installation CentraSite Registry

The CentraSite Registry service is available for download at

http://neon-project.org/ACollab/drafting/revisions.php?id=707

5.2.1 Products

It requires the following components (or equivalents) to be installed

5.2.1.1 Axis2

Download .war at http://ws.apache.org/axis2/ and install.

5.2.1.2 CentraSite Community Edition

Download at http://www.softwareag.com/Corporate/community/centrasite/default.asp

(or directly at http://www.infoq.com/zones/centrasite/download/centrasite) and install the CentraSite Server, the application tier and the Eclipse plugins.

5.2.1.3 Servlet Container Tomcat

You can use the Tomcat of the CentraSite community edition or you can download at http://tomcat.apache.org/ and install.

The Service implementation relies on the Axis2 runtime. This requires only a servlet container.

Thus you can also use the Axis2 simple server or any other application server.

5.2.2 Deployment Steps

The deployment of the NeOn Registry service CentraSite consists of two phases:

- Deploy OMV metamodel to CentraSite registry
- Deploy web service implementation into Tomcat

5.2.2.1 Import OMV Definitions into CentraSite

NeOnRegOMV\CSImpl\impl\Prerequisite\centrasite_imports

First we need to import the archive *OMV_CS_import.zip*. It contains all OMV definitions, schemas and predefined objects. Alternatively, one can import *OMV_CS_import_example_ontologies.zip*, which contains several ontologies that can be used for testing.

In Figure 5.2.1. and 5.2.2. we show how one can import these archives using the CentraSite plugin for Eclipse.

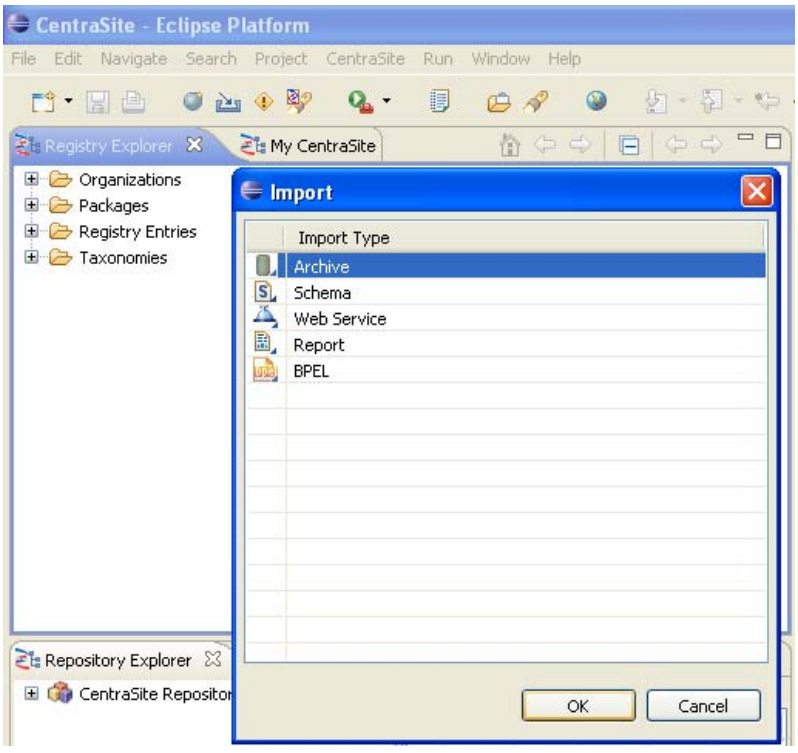


Figure 5.2.1. Deploy OMV metamodel to CentraSite

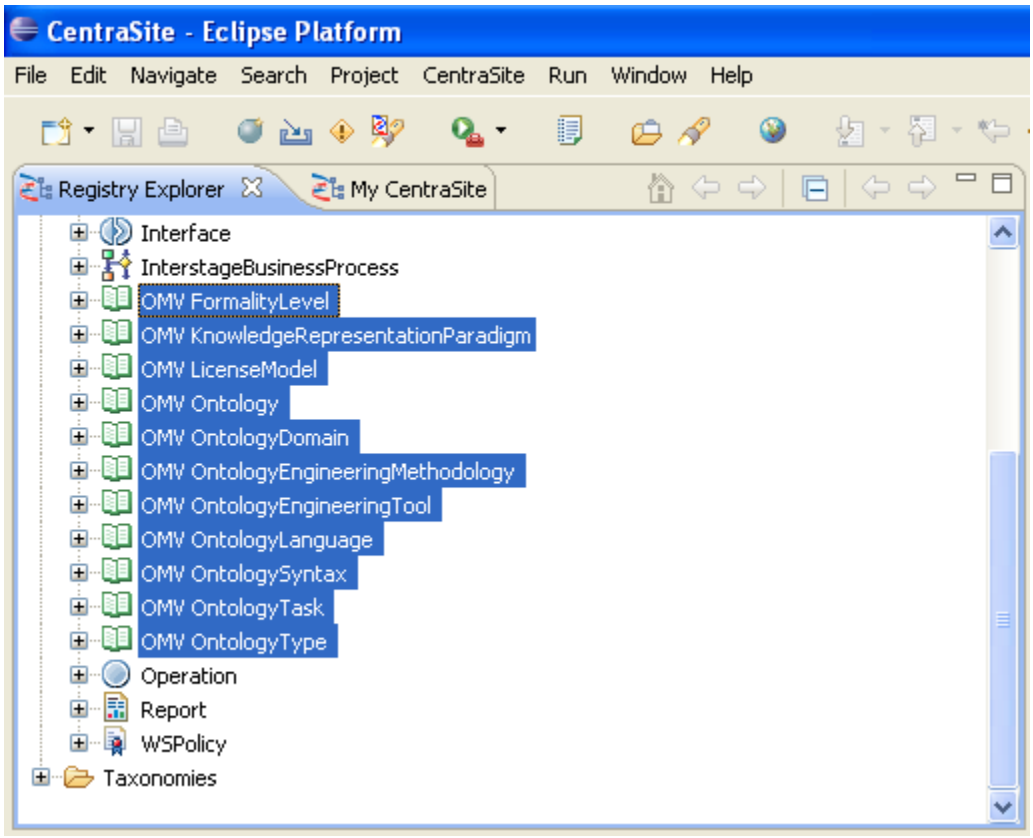


Figure 5.2.2. Registered OMV classes in CentraSite

5.2.2.2 Deploy NeOn Registry Service CentraSite into Axis2 environment of Tomcat server

The Axis2 environment is available as a .war file and can be installed on a Tomcat server. If the Axis2 simple server is used the .aar file should be copied manually to the services folder in the Axis2 installation.

The service is available as an aar-File at the following directory in the distribution:

NeOnRegOMV\CSImpl\impl\service\ebXMLRegistrySOAPService.aar

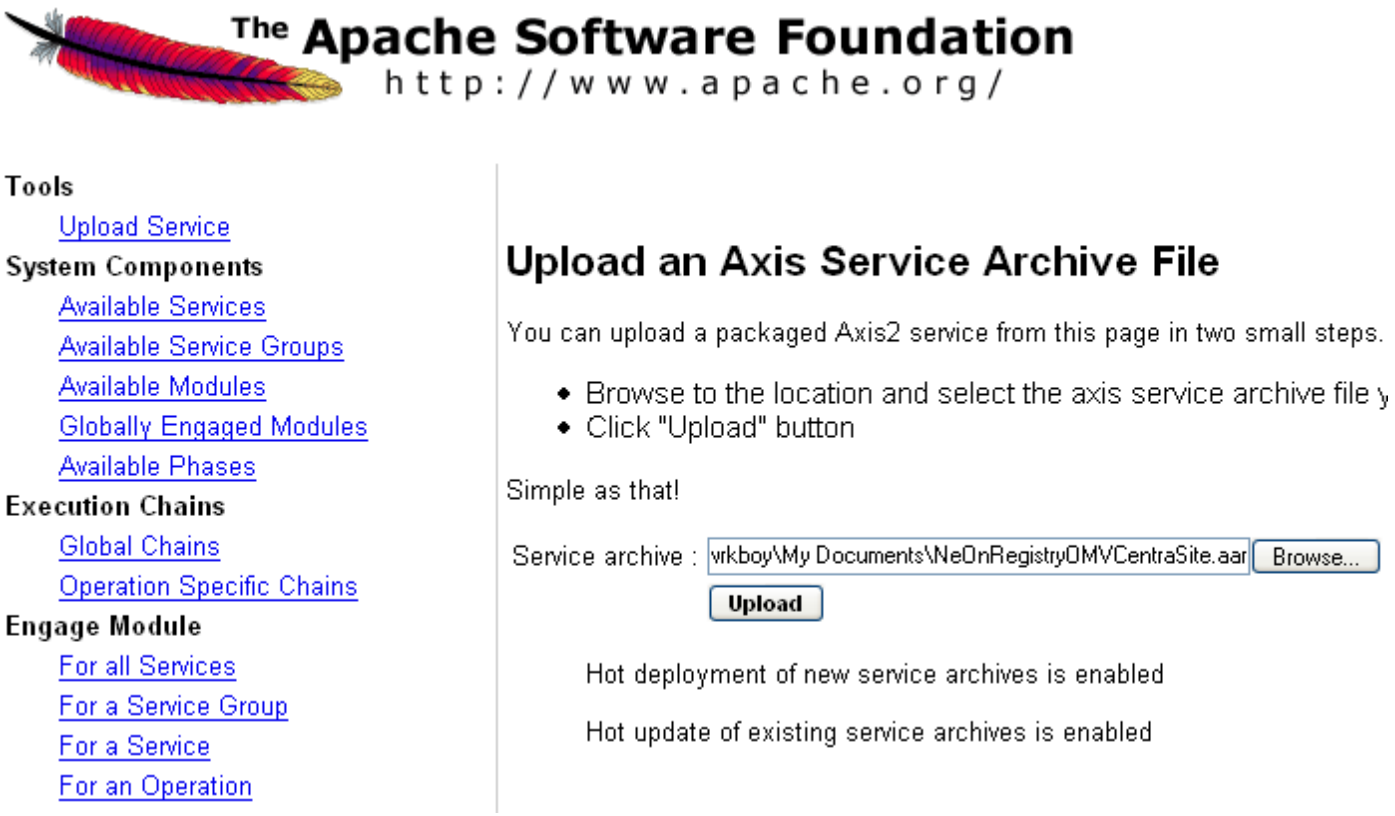


Figure 5.2.3. Axis2 Administration

The .aar file contains the service skeleton, implementation, service configuration and all necessary resources. It can be deployed in the Axis2 environment by using the Upload Service tool in the Administration Page. If hot deployment is enabled, no restart of the Tomcat server is needed (see Figure 5.2.3).

5.2.3 Verify Installation

5.2.3.1 Verify deployed service

In the Administration Page, choose “Available Services”. You should be able to see the Service **NeOnRegistryOMVCentraSite** with the following operations:

- undeprecateObjects
- submitAdhocQuery
- submitObjects
- removeObjects
- approveObjects
- updateObjects
- deprecateObjects

5.2.3.2 Fetch WSDL of service

Click on the name of the service **NeOnRegistryOMVCentraSite** to see its WSDL. Alternatively, you can copy the Service EPR and add “?wsdl” after it. Here is an example:

http://localhost:8080/axis2/services/NeOnRegistryOMVCentraSite?wsdl

The returned WSDL should match the WSDL partially included in appendix 5.3.

5.2.3.3 Example SOAP request

In order to fetch all objects of Ontology issue the following SOAP request:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:rs="urn:oasis:names:tc:ebxml-regrep:xsd:rs:3.0" xmlns:rim="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0"
xmlns:query_omv="urn:neon-project:software-ag:ebxml-omv:xsd:query-omv:2.0" xmlns:query="urn:oasis:names:tc:ebxml-regrep:xsd:query:3.0">
<SOAP-ENV:Body>
<query:AdhocQueryRequest id="http://www.test.com/test1">
<query:ResponseOption/>
<rim:AdhocQuery id="http://www.test.com/test2">
<rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-regrep:QueryLanguage:ebRSFilterQuery">
<query_omv:OntologyQuery>
</query_omv:OntologyQuery>
</rim:QueryExpression>
</rim:AdhocQuery>
</query:AdhocQueryRequest>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

delivers

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Body>
<ns4:AdhocQueryResponse xmlns:ns4="urn:oasis:names:tc:ebxml-regrep:xsd:query:3.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ns4:AdhocQueryResponse" status="urn:oasis:names:tc:ebxml-regrep:ResponseStatusType:Success" requestId="http://www.test.com/test1" startIndex="0" totalResultCount="2">
<RegistryObjectList xmlns="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0">
<ns6:Ontology xmlns:ns6="urn:neon-project:software-ag:ebxml-omv:xsd:rim-omv:2.0" xsi:type="ns6:Ontology_Type" id="uddi:ca2b9d70-4a74-11dc-8f6d-d5cb3c699a5a" acronym="Dummy Ontology" documentation="http://www.dummy.ontology.com/documentation" isOfType="uddi:a25d9a10-4a73-11dc-945a-ee2b3d8e1b8b" hasOntologySyntax="uddi:18620110-4a74-11dc-b029-ebe4068c7c64" hasOntologyLanguage="uddi:500440b0-4a74-11dc-b2ca-a31798ff0999" numClasses="5" numProperties="20" numIndividuals="200" numAxioms="4" creationDate="2007-06-05T22:00:00.000Z" modificationDate="2007-06-05T22:00:00.000Z" resourceLocator="http://www.dummy.ontology.com/resource" URI="http://www.dummy.ontology.com/" version="beta 1.0">
<Name>
<LocalizedString xsi:lang="en" charset="UTF-8" value="Dummy Ontology"/>
</Name>
<Description>
<LocalizedString xsi:lang="en" charset="UTF-8" value="Fake ontology with no meaning w.s.e"/>
</Description>
<ns6:hasCreator id="uddi:207ff1cc-25c5-544c-415c-5d98ea91060c"/>
</ns6:Ontology>
<ns6:Ontology xmlns:ns6="urn:neon-project:software-ag:ebxml-omv:xsd:rim-omv:2.0" xsi:type="ns6:Ontology_Type" id="uddi:b4546e90-4a7a-11dc-8f6d-831b46f56c19" acronym="Perfect Ontology" documentation="http://www.perfect.ontology.com/doc" isOfType="uddi:8c8b8300-4a73-11dc-82db-d71cbc84b7c6" hasOntologySyntax="uddi:1cf5ba00-4a74-11dc-aabc-861235dc2a75" hasOntologyLanguage="uddi:39d3dd00-4a74-11dc-8f89-b3bd115c0bc8" numClasses="200" numProperties="1000" numIndividuals="1000000" numAxioms="1000" creationDate="2006-06-30T22:00:00.000Z" modificationDate="2006-06-30T22:00:00.000Z" resourceLocator="http://www.perfect.ontology.com/src" URI="http://www.perfect.ontology.com/" version="Final version">
<Name>
<LocalizedString xsi:lang="en" charset="UTF-8" value="Perfect Ontology"/>
</Name>
<Description>
<LocalizedString xsi:lang="en" charset="UTF-8" value="A perfect ontology that describes it all"/>
</Description>
```

<ns6:Keywords>
<LocalizedString value="perfect"/>
</ns6:Keywords>
<ns6:hasCreator id="uddi:207ff1cc-25c5-544c-415c-5d98ea91060c"/>
<ns6:hasCreator id="uddi:66457a9a-e9df-98a7-0f91-b640b2356cdf"/>
</ns6:Ontology>
</RegistryObjectList>
</ns4:AdhocQueryResponse>
</soapenv:Body>
</soapenv:Envelope>

5.3 Appendix 3: WSDL of NeOn registry service

Here is a partially listing of the WSDL for the NeOn registry service. Most of the XML schemata have been left out.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ebXMLRegistrySOAPService" targetNamespace="urn:de.sag.centrasite.ebxml_ws" xmlns:tns="urn:de.sag.centrasite.ebxml_ws"
xmlns:lcm="urn:oasis:names:tc:ebxml-regrep:xsd:lcm:3.0" xmlns:query="urn:oasis:names:tc:ebxml-regrep:xsd:query:3.0" xmlns:ns1="urn:neon-
project:software-ag:ebxml-omv:xsd:rim-omv:2.0" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns="http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns2="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:query-omv="urn:neon-project:software-ag:ebxml-omv:xsd:query-omv:2.0"
xmlns:rs="urn:oasis:names:tc:ebxml-regrep:xsd:rs:3.0" xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <schema attributeFormDefault="unqualified" elementFormDefault="qualified" targetNamespace="urn:oasis:names:tc:ebxml-regrep:xsd:rs:3.0"
xmlns="http://www.w3.org/2001/XMLSchema" xmlns:rim="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0" xmlns:rs="urn:oasis:names:tc:ebxml-
regrep:xsd:rs:3.0">
                        <annotation>
                                <documentation xml:lang="en">The schema for OASIS ebXML Registry Services</documentation>
                        </annotation>
                        <!-- Import the rim.xsd file with XML schema mapping from RIM -->
                        <import namespace="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0"
schemaLocation="ebXMLRegistrySOAPService?xsd=xsd0.xsd"/>
                        <complexType name="RegistryRequestType">
                                <annotation>
                                        <documentation xml:lang="en">Base type for all ebXML Registry requests</documentation>
                                </annotation>
                                <sequence>
                                        <!-- every request may be extended using Slots. -->
                                        <element minOccurs="0" name="RequestSlotList" type="rim:SlotListType"/>
                                </sequence>
                                <attribute name="id" type="anyURI" use="optional"/>
                                <attribute name="comment" type="string" use="optional"/>
                                <!--Comment may be used by requestor to describe the request. Used in VersionInfo.comment-->
                        </complexType>
                        <element name="RegistryRequest" type="rs:RegistryRequestType"/>
                        <complexType name="RegistryErrorListType">
                                <sequence>
                                        <element maxOccurs="unbounded" ref="rs:RegistryError"/>
                                </sequence>
                                <attribute name="highestSeverity" type="rim:referenceURI" use="optional"/>
                        </complexType>
                        <element name="RegistryErrorList" type="rs:RegistryErrorListType">
                                <annotation>
                                        <documentation xml:lang="en">The RegistryErrorList is derived from the ErrorList element
from the ebXML Message Service Specification</documentation>
                                </annotation>
                        </element>
                        <complexType name="RegistryErrorType">
                                <simpleContent>
                                        <extension base="string">
                                                <attribute name="codeContext" type="string" use="required"/>
                                                <attribute name="errorCode" type="string" use="required"/>
                                                <attribute default="urn:oasis:names:tc:ebxml-regrep:ErrorSeverityType:Error"
name="severity" type="rim:referenceURI"/>
                                                <attribute name="location" type="string" use="optional"/>
                                        </extension>
                                </simpleContent>
                        </complexType>
                        <element name="RegistryError" type="rs:RegistryErrorType"/>
                        <complexType name="RegistryResponseType">
                                <annotation>
                                        <documentation xml:lang="en">Base type for all ebXML Registry
responses</documentation>
                                </annotation>
                                <sequence>
                                        <!-- every response may be extended using Slots. -->
                                        <element minOccurs="0" name="ResponseSlotList" type="rim:SlotListType"/>
                                        <element minOccurs="0" ref="rs:RegistryErrorList"/>
                                </sequence>
                                <attribute name="status" type="rim:referenceURI" use="required"/>
                                <attribute name="requestId" type="anyURI" use="optional"/>
                                <!-- id is the request if for the request for which this is a response -->
                        </complexType>
                        <element name="RegistryResponse" type="rs:RegistryResponseType"/>
```

```
                </schema>
   <schema attributeFormDefault="unqualified" elementFormDefault="qualified" targetNamespace="urn:oasis:names:tc:ebxml-regrep:xsd:query:3.0"
xmlns="http://www.w3.org/2001/XMLSchema" xmlns:query="urn:oasis:names:tc:ebxml-regrep:xsd:query:3.0"
xmlns:rim="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0" xmlns:rs="urn:oasis:names:tc:ebxml-regrep:xsd:rs:3.0">
                        <!-- Import the rim.xsd file with XML schema mapping from RIM -->
                        <import namespace="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0"
schemaLocation="ebXMLRegistrySOAPService?xsd=xsd0.xsd"/>

                        <simpleType name="ResponseOptionReturnType">
                                <restriction base="NCName">
                                        <enumeration value="ObjectRef"/>
                                        <enumeration value="RegistryObject"/>
                                        <enumeration value="LeafClass"/>
                                        <enumeration value="LeafClassWithRepositoryItem"/>
                                </restriction>
                        </simpleType>
                        <complexType name="ResponseOptionType">
                                <attribute default="RegistryObject" name="returnType" type="query:ResponseOptionReturnType"/>
                                <attribute default="false" name="returnComposedObjects" type="boolean"/>
                        </complexType>
                        <element name="ResponseOption" type="query:ResponseOptionType"/>
                        <element name="AdhocQueryRequest">
                                <annotation>
                                        <documentation xml:lang="en">An Ad hoc query request specifies an ad hoc
query.</documentation>
                                </annotation>
                                <complexType>
                                        <complexContent>
                                                <extension base="rs:RegistryRequestType">
                                                        <sequence>
                                                                <element ref="query:ResponseOption"/>
                                                                <element ref="rim:AdhocQuery"/>
                                                        </sequence>
                                                        <attribute default="false" name="federated" type="boolean"
use="optional"/>
                                                        <attribute name="federation" type="anyURI" use="optional"/>
                                                        <attribute default="0" name="startIndex" type="integer"/>
                                                        <attribute default="-1" name="maxResults" type="integer"/>
                                                </extension>
                                        </complexContent>
                                </complexType>
                        </element>
                        <element name="AdhocQueryResponse">
                                <annotation>
                                        <documentation xml:lang="en">
     The response includes a RegistryObjectList which has zero or more
     RegistryObjects that match the query specified in AdhocQueryRequest.
    </documentation>
                                </annotation>
                                <complexType>
                                        <complexContent>
                                                <extension base="rs:RegistryResponseType">
                                                        <sequence>
                                                                <element ref="rim:RegistryObjectList"/>
                                                        </sequence>
                                                        <attribute default="0" name="startIndex" type="integer"/>
                                                        <attribute name="totalResultCount" type="integer" use="optional"/>
                                                </extension>
                                        </complexContent>
                                </complexType>
                        </element>
                        <complexType abstract="true" name="FilterQueryType">
                                <sequence>
                                        <element minOccurs="0" name="PrimaryFilter" type="query:FilterType"/>
                                </sequence>
                        </complexType>
                        <complexType abstract="true" name="BranchType">
                                <complexContent>
                                        <extension base="query:FilterQueryType"/>
                                </complexContent>
                        </complexType>
                        <complexType name="InternationalStringBranchType">
                                <complexContent>
                                        <extension base="query:BranchType">
                                                <sequence>
```

```
                                                <element maxOccurs="unbounded" minOccurs="0"
name="LocalizedStringFilter" type="query:FilterType"/>
                                        </sequence>
                                </extension>
                        </complexContent>
                </complexType>
                <complexType name="SlotBranchType">
                        <complexContent>
                                <extension base="query:BranchType">
                                        <sequence/>
                                </extension>
                        </complexContent>
                </complexType>
                <complexType name="RegistryObjectQueryType">
                        <complexContent>
                                <extension base="query:FilterQueryType">
                                        <sequence>
                                                <element maxOccurs="unbounded" minOccurs="0" name="SlotBranch"
type="query:SlotBranchType"/>
                                                <element minOccurs="0" name="NameBranch"
type="query:InternationalStringBranchType"/>
                                                <element minOccurs="0" name="DescriptionBranch"
type="query:InternationalStringBranchType"/>
                                                <element minOccurs="0" name="VersionInfoFilter"
type="query:FilterType"/>
                                                <element maxOccurs="unbounded" minOccurs="0"
ref="query:ClassificationQuery"/>
                                                <element maxOccurs="unbounded" minOccurs="0"
ref="query:ExternalIdentifierQuery"/>
                                                <element minOccurs="0" name="ObjectTypeQuery"
type="query:ClassificationNodeQueryType"/>
                                                <element minOccurs="0" name="StatusQuery"
type="query:ClassificationNodeQueryType"/>
                                                <element maxOccurs="unbounded" minOccurs="0"
name="SourceAssociationQuery" type="query:AssociationQueryType"/>
                                                <element maxOccurs="unbounded" minOccurs="0"
name="TargetAssociationQuery" type="query:AssociationQueryType"/>
                                        </sequence>
                                </extension>
                        </complexContent>
                </complexType>
                <element name="RegistryObjectQuery" type="query:RegistryObjectQueryType"/>
                <complexType name="AssociationQueryType">
                        <complexContent>
                                <extension base="query:RegistryObjectQueryType">
                                        <sequence>
                                                <element minOccurs="0" name="AssociationTypeQuery"
type="query:ClassificationNodeQueryType"/>
                                                <element minOccurs="0" name="SourceObjectQuery"
type="query:RegistryObjectQueryType"/>
                                                <element minOccurs="0" name="TargetObjectQuery"
type="query:RegistryObjectQueryType"/>
                                        </sequence>
                                </extension>
                        </complexContent>
                </complexType>
                <element name="AssociationQuery" type="query:AssociationQueryType"/>
                <complexType name="AuditableEventQueryType">
                        <complexContent>
                                <extension base="query:RegistryObjectQueryType">
                                        <sequence>
                                                <element maxOccurs="unbounded" minOccurs="0"
name="AffectedObjectQuery" type="query:RegistryObjectQueryType"/>
                                                <element minOccurs="0" name="EventTypeQuery"
type="query:ClassificationNodeQueryType"/>
                                                <element minOccurs="0" name="UserQuery"
type="query:UserQueryType"/>
                                        </sequence>
                                </extension>
                        </complexContent>
                </complexType>
                <element name="AuditableEventQuery" type="query:AuditableEventQueryType"/>
                <complexType name="ClassificationQueryType">
                        <complexContent>
                                <extension base="query:RegistryObjectQueryType">
```

```
                                                <sequence>
                                                        <element minOccurs="0" ref="query:ClassificationSchemeQuery"/>
                                                        <element minOccurs="0" name="ClassifiedObjectQuery"
type="query:RegistryObjectQueryType"/>
                                                        <element minOccurs="0" ref="query:ClassificationNodeQuery"/>
                                                </sequence>
                                        </extension>
                                </complexContent>
                        </complexType>
                        <element name="ClassificationQuery" type="query:ClassificationQueryType"/>
                        <complexType name="ClassificationNodeQueryType">
                                <complexContent>
                                        <extension base="query:RegistryObjectQueryType">
                                                <sequence>
                                                        <element minOccurs="0" name="ParentQuery"
type="query:RegistryObjectQueryType"/>
                                                        <element maxOccurs="unbounded" minOccurs="0"
name="ChildrenQuery" type="query:ClassificationNodeQueryType"/>
                                                </sequence>
                                        </extension>
                                </complexContent>
                        </complexType>
                        <element name="ClassificationNodeQuery" type="query:ClassificationNodeQueryType"/>
                        <complexType name="ClassificationSchemeQueryType">
                                <complexContent>
                                        <extension base="query:RegistryObjectQueryType">
                                                <sequence>
                                                        <element maxOccurs="unbounded" minOccurs="0"
name="ChildrenQuery" type="query:ClassificationNodeQueryType"/>
                                                        <element minOccurs="0" name="NodeTypeQuery"
type="query:ClassificationNodeQueryType"/>
                                                </sequence>
                                        </extension>
                                </complexContent>
                        </complexType>
                        <element name="ClassificationSchemeQuery" type="query:ClassificationSchemeQueryType"/>
                        <complexType name="ExternalIdentifierQueryType">
                                <complexContent>
                                        <extension base="query:RegistryObjectQueryType">
                                                <sequence>
                                                        <element minOccurs="0" ref="query:RegistryObjectQuery"/>
                                                        <element minOccurs="0" name="IdentificationSchemeQuery"
type="query:ClassificationSchemeQueryType"/>
                                                </sequence>
                                        </extension>
                                </complexContent>
                        </complexType>
                        <element name="ExternalIdentifierQuery" type="query:ExternalIdentifierQueryType"/>
                        <complexType name="ExternalLinkQueryType">
                                <complexContent>
                                        <extension base="query:RegistryObjectQueryType">
                                                <sequence/>
                                        </extension>
                                </complexContent>
                        </complexType>
                        <element name="ExternalLinkQuery" type="query:ExternalLinkQueryType"/>
                        <complexType name="ExtrinsicObjectQueryType">
                                <complexContent>
                                        <extension base="query:RegistryObjectQueryType">
                                                <sequence>
                                                        <element minOccurs="0" name="ContentVersionInfoFilter"
type="query:FilterType"/>
                                                </sequence>
                                        </extension>
                                </complexContent>
                        </complexType>
                        <element name="ExtrinsicObjectQuery" type="query:ExtrinsicObjectQueryType"/>
                        <complexType name="OrganizationQueryType">
                                <complexContent>
                                        <extension base="query:RegistryObjectQueryType">
                                                <sequence>
                                                        <element maxOccurs="unbounded" minOccurs="0"
name="AddressFilter" type="query:FilterType"/>
                                                        <element maxOccurs="unbounded" minOccurs="0"
name="TelephoneNumberFilter" type="query:FilterType"/>
```

```
                                                <element maxOccurs="unbounded" minOccurs="0"
name="EmailAddressFilter" type="query:FilterType"/>
                                                <element minOccurs="0" name="ParentQuery"
type="query:OrganizationQueryType"/>
                                                <element maxOccurs="unbounded" minOccurs="0"
name="ChildOrganizationQuery" type="query:OrganizationQueryType"/>
                                                <element minOccurs="0" name="PrimaryContactQuery"
type="query:PersonQueryType"/>
                                        </sequence>
                                </extension>
                        </complexContent>
                </complexType>
                <element name="OrganizationQuery" type="query:OrganizationQueryType"/>
                <complexType name="RegistryPackageQueryType">
                        <complexContent>
                                <extension base="query:RegistryObjectQueryType"/>
                        </complexContent>
                </complexType>
                <element name="RegistryPackageQuery" type="query:RegistryPackageQueryType"/>
                <complexType name="ServiceQueryType">
                        <complexContent>
                                <extension base="query:RegistryObjectQueryType">
                                        <sequence>
                                                <element maxOccurs="unbounded" minOccurs="0"
ref="query:ServiceBindingQuery"/>
                                        </sequence>
                                </extension>
                        </complexContent>
                </complexType>
                <element name="ServiceQuery" type="query:ServiceQueryType"/>
                <complexType name="ServiceBindingQueryType">
                        <complexContent>
                                <extension base="query:RegistryObjectQueryType">
                                        <sequence>
                                                <element minOccurs="0" ref="query:ServiceQuery"/>
                                                <element maxOccurs="unbounded" minOccurs="0"
ref="query:SpecificationLinkQuery"/>
                                                <element minOccurs="0" name="TargetBindingQuery"
type="query:ServiceBindingQueryType"/>
                                        </sequence>
                                </extension>
                        </complexContent>
                </complexType>
                <element name="ServiceBindingQuery" type="query:ServiceBindingQueryType"/>
                <complexType name="SpecificationLinkQueryType">
                        <complexContent>
                                <extension base="query:RegistryObjectQueryType">
                                        <sequence>
                                                <element minOccurs="0" name="UsageDescriptionBranch"
type="query:InternationalStringBranchType"/>
                                                <element minOccurs="0" ref="query:ServiceBindingQuery"/>
                                                <element minOccurs="0" name="SpecificationObjectQuery"
type="query:RegistryObjectQueryType"/>
                                        </sequence>
                                </extension>
                        </complexContent>
                </complexType>
                <element name="SpecificationLinkQuery" type="query:SpecificationLinkQueryType"/>
                <complexType name="PersonQueryType">
                        <complexContent>
                                <extension base="query:RegistryObjectQueryType">
                                        <sequence>
                                                <element maxOccurs="unbounded" minOccurs="0"
name="AddressFilter" type="query:FilterType"/>
                                                <element minOccurs="0" name="PersonNameFilter"
type="query:FilterType"/>
                                                <element maxOccurs="unbounded" minOccurs="0"
name="TelephoneNumberFilter" type="query:FilterType"/>
                                                <element maxOccurs="unbounded" minOccurs="0"
name="EmailAddressFilter" type="query:FilterType"/>
                                        </sequence>
                                </extension>
                        </complexContent>
                </complexType>
                <element name="PersonQuery" type="query:PersonQueryType"/>
```

```
                    <complexType name="UserQueryType">
                        <complexContent>
                            <extension base="query:PersonQueryType">
                                <sequence/>
                            </extension>
                        </complexContent>
                    </complexType>
                    <element name="UserQuery" type="query:UserQueryType"/>
                    <complexType name="RegistryQueryType">
                        <complexContent>
                            <extension base="query:RegistryObjectQueryType">
                                <sequence>
                                    <element minOccurs="0" name="OperatorQuery"
type="query:OrganizationQueryType"/>
                                </sequence>
                            </extension>
                        </complexContent>
                    </complexType>
                    <element name="RegistryQuery" type="query:RegistryQueryType"/>
                    <complexType name="FederationQueryType">
                        <complexContent>
                            <extension base="query:RegistryObjectQueryType">
                                <sequence/>
                            </extension>
                        </complexContent>
                    </complexType>
                    <element name="FederationQuery" type="query:FederationQueryType"/>
                    <complexType name="AdhocQueryQueryType">
                        <complexContent>
                            <extension base="query:RegistryObjectQueryType">
                                <sequence>
                                    <element minOccurs="0" name="QueryExpressionBranch"
type="query:QueryExpressionBranchType"/>
                                </sequence>
                            </extension>
                        </complexContent>
                    </complexType>
                    <element name="AdhocQueryQuery" type="query:AdhocQueryQueryType"/>
                    <complexType name="QueryExpressionBranchType">
                        <complexContent>
                            <extension base="query:BranchType">
                                <sequence>
                                    <element minOccurs="0" name="QueryLanguageQuery"
type="query:ClassificationNodeQueryType"/>
                                </sequence>
                            </extension>
                        </complexContent>
                    </complexType>
                    <complexType name="NotificationQueryType">
                        <complexContent>
                            <extension base="query:RegistryObjectQueryType">
                                <sequence>
                                    <element minOccurs="0" ref="query:RegistryObjectQuery"/>
                                </sequence>
                            </extension>
                        </complexContent>
                    </complexType>
                    <element name="NotificationQuery" type="query:NotificationQueryType"/>
                    <complexType name="SubscriptionQueryType">
                        <complexContent>
                            <extension base="query:RegistryObjectQueryType">
                                <sequence>
                                    <element minOccurs="0" name="SelectorQuery"
type="query:AdhocQueryQueryType"/>
                                </sequence>
                            </extension>
                        </complexContent>
                    </complexType>
                    <element name="SubscriptionQuery" type="query:SubscriptionQueryType"/>
                    <!-- The Filter type hierarchy -->
                    <complexType name="FilterType">
                        <attribute default="false" name="negate" type="boolean"/>
                    </complexType>
                    <element abstract="true" name="Filter" type="query:FilterType"/>
                    <simpleType name="LogicalOperatorType">
```

```
                        <restriction base="NCName">
                                <enumeration value="AND"/>
                                <enumeration value="OR"/>
                        </restriction>
                </simpleType>
                <complexType name="CompoundFilterType">
                        <complexContent>
                                <extension base="query:FilterType">
                                        <sequence>
                                                <element name="LeftFilter" type="query:FilterType"/>
                                                <element name="RightFilter" type="query:FilterType"/>
                                        </sequence>
                                        <attribute name="logicalOperator" type="query:LogicalOperatorType"
use="required"/>
                                </extension>
                        </complexContent>
                </complexType>
                <element name="CompoundFilter" type="query:CompoundFilterType"/>
                <simpleType name="ComparatorType">
                        <restriction base="NCName">
                                <enumeration value="LE"/>
                                <enumeration value="LT"/>
                                <enumeration value="GE"/>
                                <enumeration value="GT"/>
                                <enumeration value="EQ"/>
                                <enumeration value="NE"/>
                                <enumeration value="Like"/>
                                <enumeration value="NotLike"/>
                        </restriction>
                </simpleType>
                <complexType abstract="true" name="SimpleFilterType">
                        <complexContent>
                                <extension base="query:FilterType">
                                        <attribute name="domainAttribute" type="string" use="required"/>
                                        <attribute name="comparator" type="query:ComparatorType" use="required"/>
                                </extension>
                        </complexContent>
                </complexType>
                <complexType name="BooleanFilterType">
                        <complexContent>
                                <extension base="query:SimpleFilterType">
                                        <attribute name="value" type="boolean" use="required"/>
                                </extension>
                        </complexContent>
                </complexType>
                <element name="BooleanFilter" type="query:BooleanFilterType"/>
                <complexType name="IntegerFilterType">
                        <complexContent>
                                <extension base="query:SimpleFilterType">
                                        <attribute name="value" type="integer" use="required"/>
                                </extension>
                        </complexContent>
                </complexType>
                <element name="IntegerFilter" type="query:IntegerFilterType"/>
                <complexType name="FloatFilterType">
                        <complexContent>
                                <extension base="query:SimpleFilterType">
                                        <attribute name="value" type="float" use="required"/>
                                </extension>
                        </complexContent>
                </complexType>
                <element name="FloatFilter" type="query:FloatFilterType"/>
                <complexType name="DateTimeFilterType">
                        <complexContent>
                                <extension base="query:SimpleFilterType">
                                        <attribute name="value" type="dateTime" use="required"/>
                                </extension>
                        </complexContent>
                </complexType>
                <element name="DateTimeFilter" type="query:DateTimeFilterType"/>
                <complexType name="StringFilterType">
                        <complexContent>
                                <extension base="query:SimpleFilterType">
                                        <attribute name="value" type="string" use="required"/>
                                </extension>
```

```
                    </complexContent>
                </complexType>
                <element name="StringFilter" type="query:StringFilterType"/>
            </schema>
    <schema attributeFormDefault="unqualified" elementFormDefault="qualified" targetNamespace="urn:oasis:names:tc:ebxml-regrep:xsd:lcm:3.0"
xmlns="http://www.w3.org/2001/XMLSchema" xmlns:lcm="urn:oasis:names:tc:ebxml-regrep:xsd:lcm:3.0" xmlns:rim="urn:oasis:names:tc:ebxml-
regrep:xsd:rim:3.0" xmlns:rs="urn:oasis:names:tc:ebxml-regrep:xsd:rs:3.0">
                <annotation>
                    <documentation xml:lang="en">The schema for OASIS ebXML Registry Services</documentation>
                </annotation>
                <!-- Import the rim.xsd file with XML schema mapping from RIM -->
                <import namespace="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0"
schemaLocation="ebXMLRegistrySOAPService?xsd=xsd0.xsd"/>

                <element name="SubmitObjectsRequest">
                    <annotation>
                        <documentation xml:lang="en">The SubmitObjectsRequest allows one to submit a list of
RegistryObject elements. Each RegistryEntry element provides metadata for a single submitted object.  Note that the repository item being submitted
is in a separate document that is not in this DTD. The ebXML Messaging Services Specfication defines packaging, for submission, of the metadata of
a repository item with the repository item itself. The value of the id attribute of the ExtrinsicObject element must be the same as the xlink:href
attribute within the Reference element within the Manifest element of the MessageHeader.</documentation>
                    </annotation>
                    <complexType>
                        <complexContent>
                            <extension base="rs:RegistryRequestType">
                                <sequence>
                                    <element ref="rim:RegistryObjectList"/>
                                </sequence>
                            </extension>
                        </complexContent>
                    </complexType>
                </element>
                <element name="UpdateObjectsRequest">
                    <annotation>
                        <documentation xml:lang="en">The UpdateObjectsRequest allows one to update a list of
RegistryObject elements. Each RegistryEntry element provides metadata for a single submitted object.  Note that the repository item being submitted
is in a separate document that is not in this DTD. The ebXML Messaging Services Specfication defines packaging, for submission, of the metadata of
a repository item with the repository item itself. The value of the id attribute of the ExtrinsicObject element must be the same as the xlink:href
attribute within the Reference element within the Manifest element of the MessageHeader.</documentation>
                    </annotation>
                    <complexType>
                        <complexContent>
                            <extension base="rs:RegistryRequestType">
                                <sequence>
                                    <element ref="rim:RegistryObjectList"/>
                                </sequence>
                            </extension>
                        </complexContent>
                    </complexType>
                </element>
                <element name="ApproveObjectsRequest">
                    <annotation>
                        <documentation xml:lang="en">
    The ObjectRefList and AdhocQuery identify the list of
    objects being approved.
  </documentation>
                    </annotation>
                    <complexType>
                        <complexContent>
                            <extension base="rs:RegistryRequestType">
                                <sequence>
                                    <element minOccurs="0" ref="rim:AdhocQuery"/>
                                    <element minOccurs="0" ref="rim:ObjectRefList"/>
                                </sequence>
                            </extension>
                        </complexContent>
                    </complexType>
                </element>
                <element name="DeprecateObjectsRequest">
                    <annotation>
                        <documentation xml:lang="en">
    The ObjectRefList and AdhocQuery identify the list of
    objects being deprecated.
  </documentation>
                    </annotation>
```

```
                                        <complexType>
                                                <complexContent>
                                                        <extension base="rs:RegistryRequestType">
                                                                <sequence>
                                                                        <element minOccurs="0" ref="rim:AdhocQuery"/>
                                                                        <element minOccurs="0" ref="rim:ObjectRefList"/>
                                                                </sequence>
                                                        </extension>
                                                </complexContent>
                                        </complexType>
                                </element>
                                <element name="UndeprecateObjectsRequest">
                                        <annotation>
                                                <documentation xml:lang="en">
    The ObjectRefList is the list of
    refs to the registry entries being un-deprecated.
  </documentation>
                                        </annotation>
                                        <complexType>
                                                <complexContent>
                                                        <extension base="rs:RegistryRequestType">
                                                                <sequence>
                                                                        <element minOccurs="0" ref="rim:AdhocQuery"/>
                                                                        <element minOccurs="0" ref="rim:ObjectRefList"/>
                                                                </sequence>
                                                        </extension>
                                                </complexContent>
                                        </complexType>
                                </element>
                                <element name="RemoveObjectsRequest">
                                        <annotation>
                                                <documentation xml:lang="en">
    The ObjectRefList is the list of
    refs to the registry entrys being removed
  </documentation>
                                        </annotation>
                                        <complexType>
                                                <complexContent>
                                                        <extension base="rs:RegistryRequestType">
                                                                <sequence>
                                                                        <element minOccurs="0" ref="rim:AdhocQuery"/>
                                                                        <element minOccurs="0" ref="rim:ObjectRefList"/>
                                                                </sequence>
                                                                <attribute default="urn:oasis:names:tc:ebxml-
regrep:DeletionScopeType:DeleteAll" name="deletionScope" type="rim:referenceURI" use="optional"/>
                                                        </extension>
                                                </complexContent>
                                        </complexType>
                                </element>
                                <element name="RelocateObjectsRequest">
                                        <complexType>
                                                <complexContent>
                                                        <extension base="rs:RegistryRequestType">
                                                                <sequence>
                                                                        <element ref="rim:AdhocQuery"/>
                                                                        <element name="SourceRegistry"
type="rim:ObjectRefType"/>
                                                                        <element name="DestinationRegistry"
type="rim:ObjectRefType"/>
                                                                        <element name="OwnerAtSource"
type="rim:ObjectRefType"/>
                                                                        <element name="OwnerAtDestination"
type="rim:ObjectRefType"/>
                                                                </sequence>
                                                        </extension>
                                                </complexContent>
                                        </complexType>
                                </element>
                                <element name="AcceptObjectsRequest">
                                        <!-- The ObjectRefList must only contain local ObjectRefs such that they do not specify home attribute --
>
                                        <complexType>
                                                <complexContent>
                                                        <extension base="rs:RegistryRequestType">
                                                                <attribute name="correlationId" type="anyURI" use="required"/>
```

```
                                        </extension>
                                    </complexContent>
                                </complexType>
                            </element>
                        </schema>
    <schema attributeFormDefault="unqualified" elementFormDefault="qualified" targetNamespace="urn:neon-project:software-ag:ebxml-
omv:xsd:rim-omv:2.0" xmlns="http://www.w3.org/2001/XMLSchema" xmlns:rim="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0"
xmlns:tns="urn:neon-project:software-ag:ebxml-omv:xsd:rim-omv:2.0">
                            <annotation>
                                <documentation xml:lang="en">The schema for NEON OMV 2.0 extention of the OASIS ebXML
Registry Information Model</documentation>
                            </annotation>
                            <import namespace="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0"
schemaLocation="ebXMLRegistrySOAPService?xsd=xsd0.xsd"/>
                            <!-- Begin information model mapping from ebRIM. -->
                            <complexType name="OMVObjectRefType">
                                <annotation>
                                    <documentation xml:lang="en">A type that would allow referencing multiply objects within
an OMV ontology.</documentation>
                                </annotation>
                                <attribute name="id" type="rim:referenceURI" use="required"/>
                            </complexType>
                            <complexType name="OMVRegistryObjectType">
                                <annotation>
                                    <documentation xml:lang="en">Basic OMV type that adds the acronym and documentation
elements to the ebXML RegistryObject.</documentation>
                                </annotation>
                                <complexContent>
                                    <extension base="rim:RegistryObjectType">
                                        <attribute name="acronym" type="rim:ShortName" use="required"/>
                                        <attribute name="documentation" type="anyURI" use="optional"/>
                                    </extension>
                                </complexContent>
                            </complexType>
                            <complexType name="Ontology_Type">
                                <annotation>
                                    <documentation xml:lang="en">OMV Ontology type.</documentation>
                                </annotation>
                                <complexContent>
                                    <extension base="tns:OMVRegistryObjectType">
                                        <sequence minOccurs="0">
                                            <element maxOccurs="unbounded" minOccurs="0" name="Keywords"
type="rim:InternationalStringType"/>
                                            <element maxOccurs="unbounded" minOccurs="0"
name="NaturalLanguage" type="rim:InternationalStringType"/>
                                            <!--hasContributor value MUST reference a ebXML User or
Organization -->
                                            <element maxOccurs="unbounded" minOccurs="0"
name="hasContributor" type="tns:OMVObjectRefType"/>
                                            <!--hasCreator value MUST reference a ebXML User or Organization --
>
                                            <element maxOccurs="unbounded" name="hasCreator"
type="tns:OMVObjectRefType"/>
                                            <!--usedOntologyEngineeringTool value MUST reference an
OntologyEngineeringTool -->
                                            <element maxOccurs="unbounded" minOccurs="0"
name="usedOntologyEngineeringTool" type="tns:OMVObjectRefType"/>
                                            <!--usedOntologyEngineeringMethodology value MUST reference an
OntologyEngineeringMethodology -->
                                            <element maxOccurs="unbounded" minOccurs="0"
name="usedOntologyEngineeringMethodology" type="tns:OMVObjectRefType"/>
                                            <!--usedKnowledgeRepresentationParadigm value MUST reference a
KnowledgeRepresentationParadigm -->
                                            <element maxOccurs="unbounded" minOccurs="0"
name="usedKnowledgeRepresentationParadigm" type="tns:OMVObjectRefType"/>
                                            <!--designedForOntologyTask value MUST reference an OntologyTask
-->
                                            <element maxOccurs="unbounded" minOccurs="0"
name="designedForOntologyTask" type="tns:OMVObjectRefType"/>
                                            <!--useImports value MUST reference an Ontology -->
                                            <element maxOccurs="unbounded" minOccurs="0" name="useImports"
type="tns:OMVObjectRefType"/>
                                            <!--isBackwardCompatibleWith value MUST reference an Ontology -->
                                            <element maxOccurs="unbounded" minOccurs="0"
name="isBackwardCompatibleWith" type="tns:OMVObjectRefType"/>
```

```
                                                <!--isIncompatibleWith value MUST reference an Ontology -->
                                                <element maxOccurs="unbounded" minOccurs="0"
name="isIncompatibleWith" type="tns:OMVObjectRefType"/>
                                        </sequence>
                                        <attribute name="isOfType" type="rim:referenceURI" use="required"/>
                                        <attribute name="hasOntologySyntax" type="rim:referenceURI" use="required"/>
                                        <attribute name="hasOntologyLanguage" type="rim:referenceURI"
use="required"/>
                                        <attribute name="hasLicense" type="rim:referenceURI" use="optional"/>
                                        <attribute name="hasFormalityLevel" type="rim:referenceURI" use="optional"/>
                                        <attribute name="hasPriorVersion" type="rim:referenceURI" use="optional"/>
                                        <attribute name="numClasses" type="integer" use="required"/>
                                        <attribute name="numProperties" type="integer" use="required"/>
                                        <attribute name="numIndividuals" type="integer" use="required"/>
                                        <attribute name="numAxioms" type="integer" use="required"/>
                                        <attribute name="ontologyStatus" type="rim:ShortName" use="optional"/>
                                        <attribute name="creationDate" type="dateTime" use="required"/>
                                        <attribute name="modificationDate" type="dateTime" use="optional"/>
                                        <attribute name="resourceLocator" type="anyURI" use="required"/>
                                        <attribute name="URI" type="anyURI" use="required"/>
                                        <attribute name="version" type="rim:FreeFormText" use="required"/>
                                </extension>
                                <!--isOfType value MUST reference an OntologyType -->
                                <!--hasOntologySyntax value MUST reference an OntologySyntax -->
                                <!--hasOntologyLanguage value MUST reference an OntologyLanguage -->
                                <!--hasLicense value MUST reference a LicenseModel -->
                                <!--hasFormalityLevel value MUST reference a FormalityLevel -->
                                <!--hasPriorVersion value MUST reference an Ontology -->
                        </complexContent>
                </complexType>
                <element name="Ontology" substitutionGroup="rim:Identifiable" type="tns:Ontology_Type"/>
                <complexType name="OntologyType_Type">
                        <annotation>
                                <documentation xml:lang="en">OMV OntologyType type.</documentation>
                        </annotation>
                        <complexContent>
                                <extension base="tns:OMVRegistryObjectType">
                                        <sequence minOccurs="0">
                                                <!--definedBy value MUST reference a ebXML User or Organization --
>
                                                <element maxOccurs="unbounded" minOccurs="0" name="definedBy"
type="tns:OMVObjectRefType"/>
                                        </sequence>
                                </extension>
                        </complexContent>
                </complexType>
                <element name="OntologyType" substitutionGroup="rim:Identifiable" type="tns:OntologyType_Type"/>
                <complexType name="LicenseModelType">
                        <annotation>
                                <documentation xml:lang="en">OMV LicenseModel type.</documentation>
                        </annotation>
                        <complexContent>
                                <extension base="tns:OMVRegistryObjectType">
                                        <sequence minOccurs="0">
                                                <!--specifiedBy value MUST reference a ebXML User or Organization -
->
                                                <element maxOccurs="unbounded" minOccurs="0"
name="specifiedBy" type="tns:OMVObjectRefType"/>
                                        </sequence>
                                </extension>
                        </complexContent>
                </complexType>
                <element name="LicenseModel" substitutionGroup="rim:Identifiable" type="tns:LicenseModelType"/>
                <complexType name="OntologyEngineeringMethodologyType">
                        <annotation>
                                <documentation xml:lang="en">OMV OntologyEngineeringMethodology
type.</documentation>
                        </annotation>
                        <complexContent>
                                <extension base="tns:OMVRegistryObjectType">
                                        <sequence minOccurs="0">
                                                <!--developedBy value MUST reference a ebXML User or Organization
-->
                                                <element maxOccurs="unbounded" minOccurs="0"
name="developedBy" type="tns:OMVObjectRefType"/>
```

```
                                        </sequence>
                                    </extension>
                                </complexContent>
                            </complexType>
                            <element name="OntologyEngineeringMethodology" substitutionGroup="rim:Identifiable"
type="tns:OntologyEngineeringMethodologyType"/>
                            <complexType name="OntologyEngineeringToolType">
                                <annotation>
                                    <documentation xml:lang="en">OMV OntologyEngineeringTool type.</documentation>
                                </annotation>
                                <complexContent>
                                    <extension base="tns:OMVRegistryObjectType">
                                        <sequence minOccurs="0">
                                            <!--developedBy value MUST reference a ebXML User or Organization
-->
                                            <element maxOccurs="unbounded" minOccurs="0"
name="developedBy" type="tns:OMVObjectRefType"/>
                                        </sequence>
                                    </extension>
                                </complexContent>
                            </complexType>
                            <element name="OntologyEngineeringTool" substitutionGroup="rim:Identifiable"
type="tns:OntologyEngineeringToolType"/>
                            <complexType name="OntologySyntaxType">
                                <annotation>
                                    <documentation xml:lang="en">OMV OntologySyntax type.</documentation>
                                </annotation>
                                <complexContent>
                                    <extension base="tns:OMVRegistryObjectType">
                                        <sequence minOccurs="0">
                                            <!--developedBy value MUST reference a ebXML User or Organization
-->
                                            <element maxOccurs="unbounded" minOccurs="0"
name="developedBy" type="tns:OMVObjectRefType"/>
                                        </sequence>
                                    </extension>
                                </complexContent>
                            </complexType>
                            <element name="OntologySyntax" substitutionGroup="rim:Identifiable" type="tns:OntologySyntaxType"/>
                            <complexType name="OntologyLanguageType">
                                <annotation>
                                    <documentation xml:lang="en">OMV OntologyLanguage type.</documentation>
                                </annotation>
                                <complexContent>
                                    <extension base="tns:OMVRegistryObjectType">
                                        <sequence minOccurs="0">
                                            <!--developedBy value MUST reference a ebXML User or Organization
-->
                                            <element maxOccurs="unbounded" minOccurs="0"
name="developedBy" type="tns:OMVObjectRefType"/>
                                            <!--supportsRepresentationParadigm value MUST reference a
KnowledgeRepresentationParadigm -->
                                            <element maxOccurs="unbounded" minOccurs="0"
name="supportsRepresentationParadigm" type="tns:OMVObjectRefType"/>
                                            <!--hasSyntax value MUST reference an OntologySyntax -->
                                            <element maxOccurs="unbounded" minOccurs="0" name="hasSyntax"
type="tns:OMVObjectRefType"/>
                                        </sequence>
                                    </extension>
                                </complexContent>
                            </complexType>
                            <element name="OntologyLanguage" substitutionGroup="rim:Identifiable" type="tns:OntologyLanguageType"/>
                            <complexType name="KnowledgeRepresentationParadigmType">
                                <annotation>
                                    <documentation xml:lang="en">OMV KnowledgeRepresentationParadigm
type.</documentation>
                                </annotation>
                                <complexContent>
                                    <extension base="tns:OMVRegistryObjectType">
                                        <sequence minOccurs="0">
                                            <!--specifiedBy value MUST reference a ebXML User or Organization -
->
                                            <element maxOccurs="unbounded" minOccurs="0"
name="specifiedBy" type="tns:OMVObjectRefType"/>
                                        </sequence>
```

```
                                        </extension>
                                </complexContent>
                        </complexType>
                        <element name="KnowledgeRepresentationParadigm" substitutionGroup="rim:Identifiable"
type="tns:KnowledgeRepresentationParadigmType"/>
                        <complexType name="FormalityLevelType">
                                <annotation>
                                        <documentation xml:lang="en">OMV FormalityLevel type.</documentation>
                                </annotation>
                                <complexContent>
                                        <extension base="tns:OMVRegistryObjectType">
                                                <sequence minOccurs="0">
                                                        <!--specifiedBy value MUST reference a ebXML User or Organization -
->
                                                        <element maxOccurs="unbounded" minOccurs="0"
name="specifiedBy" type="tns:OMVObjectRefType"/>
                                                </sequence>
                                        </extension>
                                </complexContent>
                        </complexType>
                        <element name="FormalityLevel" substitutionGroup="rim:Identifiable" type="tns:FormalityLevelType"/>
                        <complexType name="OntologyTaskType">
                                <annotation>
                                        <documentation xml:lang="en">OMV OntologyTask type.</documentation>
                                </annotation>
                                <complexContent>
                                        <extension base="tns:OMVRegistryObjectType">
                                                <sequence minOccurs="0">
                                                        <!--specifiedBy value MUST reference a ebXML User or Organization -
->
                                                        <element maxOccurs="unbounded" minOccurs="0"
name="specifiedBy" type="tns:OMVObjectRefType"/>
                                                </sequence>
                                        </extension>
                                </complexContent>
                        </complexType>
                        <element name="OntologyTask" substitutionGroup="rim:Identifiable" type="tns:OntologyTaskType"/>
                </schema>
   <schema attributeFormDefault="unqualified" elementFormDefault="qualified" targetNamespace="urn:neon-project:software-ag:ebxml-
omv:xsd:query-omv:2.0" xmlns="http://www.w3.org/2001/XMLSchema" xmlns:query="urn:oasis:names:tc:ebxml-regrep:xsd:query:3.0"
xmlns:query_omv="urn:neon-project:software-ag:ebxml-omv:xsd:query-omv:2.0">
                        <complexType name="FormalityLevelQueryType">
                                <complexContent>
                                        <extension base="query:RegistryObjectQueryType">
                                                <sequence>
                                                        <element maxOccurs="unbounded" minOccurs="0"
name="SpecifiedByOrganizationQuery" type="query:OrganizationQueryType"/>
                                                        <element maxOccurs="unbounded" minOccurs="0"
name="SpecifiedByPersonQuery" type="query:PersonQueryType"/>
                                                </sequence>
                                        </extension>
                                </complexContent>
                        </complexType>
                        <element name="FormalityLevelQuery" type="query_omv:FormalityLevelQueryType"/>
                        <complexType name="KnowledgeRepresentationParadigmQueryType">
                                <complexContent>
                                        <extension base="query:RegistryObjectQueryType">
                                                <sequence>
                                                        <element maxOccurs="unbounded" minOccurs="0"
name="SpecifiedByOrganizationQuery" type="query:OrganizationQueryType"/>
                                                        <element maxOccurs="unbounded" minOccurs="0"
name="SpecifiedByPersonQuery" type="query:PersonQueryType"/>
                                                </sequence>
                                        </extension>
                                </complexContent>
                        </complexType>
                        <element name="KnowledgeRepresentationParadigmQuery"
type="query_omv:KnowledgeRepresentationParadigmQueryType"/>
                        <complexType name="LicenseModelQueryType">
                                <complexContent>
                                        <extension base="query:RegistryObjectQueryType">
                                                <sequence>
                                                        <element maxOccurs="unbounded" minOccurs="0"
name="SpecifiedByOrganizationQuery" type="query:OrganizationQueryType"/>
```

```
                                                <element maxOccurs="unbounded" minOccurs="0"
name="SpecifiedByPersonQuery" type="query:PersonQueryType"/>
                                            </sequence>
                                        </extension>
                                    </complexContent>
                                </complexType>
                                <element name="LicenseModelQuery" type="query_omv:LicenseModelQueryType"/>
                                <complexType name="OntologyEngineeringMethodologyQueryType">
                                    <complexContent>
                                        <extension base="query:RegistryObjectQueryType">
                                            <sequence>
                                                <element maxOccurs="unbounded" minOccurs="0"
name="DevelopedByOrganizationQuery" type="query:OrganizationQueryType"/>
                                                <element maxOccurs="unbounded" minOccurs="0"
name="DevelopedByPersonQuery" type="query:PersonQueryType"/>
                                            </sequence>
                                        </extension>
                                    </complexContent>
                                </complexType>
                                <element name="OntologyEngineeringMethodologyQuery"
type="query_omv:OntologyEngineeringMethodologyQueryType"/>
                                <complexType name="OntologyEngineeringToolQueryType">
                                    <complexContent>
                                        <extension base="query:RegistryObjectQueryType">
                                            <sequence>
                                                <element maxOccurs="unbounded" minOccurs="0"
name="DevelopedByOrganizationQuery" type="query:OrganizationQueryType"/>
                                                <element maxOccurs="unbounded" minOccurs="0"
name="DevelopedByPersonQuery" type="query:PersonQueryType"/>
                                            </sequence>
                                        </extension>
                                    </complexContent>
                                </complexType>
                                <element name="OntologyEngineeringToolQuery" type="query_omv:OntologyEngineeringToolQueryType"/>
                                <complexType name="OntologyLanguageQueryType">
                                    <complexContent>
                                        <extension base="query:RegistryObjectQueryType">
                                            <sequence>
                                                <element maxOccurs="unbounded" minOccurs="0"
name="DevelopedByOrganizationQuery" type="query:OrganizationQueryType"/>
                                                <element maxOccurs="unbounded" minOccurs="0"
name="DevelopedByPersonQuery" type="query:PersonQueryType"/>
                                                <element maxOccurs="unbounded" minOccurs="0"
name="SupportsRepresentationParadigmQuery" type="query_omv:KnowledgeRepresentationParadigmQueryType"/>
                                                <element maxOccurs="unbounded" minOccurs="0"
name="HasSyntaxQuery" type="query_omv:OntologySyntaxQueryType"/>
                                            </sequence>
                                        </extension>
                                    </complexContent>
                                </complexType>
                                <element name="OntologyLanguageQuery" type="query_omv:OntologyLanguageQueryType"/>
                                <complexType name="OntologySyntaxQueryType">
                                    <complexContent>
                                        <extension base="query:RegistryObjectQueryType">
                                            <sequence>
                                                <element maxOccurs="unbounded" minOccurs="0"
name="DevelopedByOrganizationQuery" type="query:OrganizationQueryType"/>
                                                <element maxOccurs="unbounded" minOccurs="0"
name="DevelopedByPersonQuery" type="query:PersonQueryType"/>
                                            </sequence>
                                        </extension>
                                    </complexContent>
                                </complexType>
                                <element name="OntologySyntaxQuery" type="query_omv:OntologySyntaxQueryType"/>
                                <complexType name="OntologyTaskQueryType">
                                    <complexContent>
                                        <extension base="query:RegistryObjectQueryType">
                                            <sequence>
                                                <element maxOccurs="unbounded" minOccurs="0"
name="SpecifiedByOrganizationQuery" type="query:OrganizationQueryType"/>
                                                <element maxOccurs="unbounded" minOccurs="0"
name="SpecifiedByPersonQuery" type="query:PersonQueryType"/>
                                            </sequence>
                                        </extension>
                                    </complexContent>
```

```
                    </complexType>
                    <element name="OntologyTaskQuery" type="query_omv:OntologyTaskQueryType"/>
                    <complexType name="OntologyTypeQueryType">
                        <complexContent>
                            <extension base="query:RegistryObjectQueryType">
                                <sequence>
                                    <element maxOccurs="unbounded" minOccurs="0"
name="DefinedByOrganizationQuery" type="query:OrganizationQueryType"/>
                                    <element maxOccurs="unbounded" minOccurs="0"
name="DefinedByPersonQuery" type="query:PersonQueryType"/>
                                </sequence>
                            </extension>
                        </complexContent>
                    </complexType>
                    <element name="OntologyTypeQuery" type="query_omv:OntologyTypeQueryType"/>
                    <complexType name="OntologyQueryType">
                        <complexContent>
                            <extension base="query:RegistryObjectQueryType">
                                <sequence>
                                    <element minOccurs="0" name="KeywordsBranch"
type="query:InternationalStringBranchType"/>
                                    <element maxOccurs="unbounded" minOccurs="0"
name="NaturalLanguageBranch" type="query:InternationalStringBranchType"/>
                                    <element minOccurs="0" name="IsOfTypeQuery"
type="query_omv:OntologyTypeQueryType"/>
                                    <element minOccurs="0" name="HasOntologySyntaxQuery"
type="query_omv:OntologySyntaxQueryType"/>
                                    <element minOccurs="0" name="HasOntologyLanguageQuery"
type="query_omv:OntologyLanguageQueryType"/>
                                    <element minOccurs="0" name="HasLicenseQuery"
type="query_omv:LicenseModelQueryType"/>
                                    <element minOccurs="0" name="HasFormalityLevelQuery"
type="query_omv:FormalityLevelQueryType"/>
                                    <element minOccurs="0" name="HasPriorVersionQuery"
type="query_omv:OntologyQueryType"/>
                                    <element maxOccurs="unbounded" minOccurs="0"
name="HasCreatorOrganizationQuery" type="query:OrganizationQueryType"/>
                                    <element maxOccurs="unbounded" minOccurs="0"
name="HasCreatorPersonQuery" type="query:PersonQueryType"/>
                                    <element maxOccurs="unbounded" minOccurs="0"
name="HasContributorOrganizationQuery" type="query:OrganizationQueryType"/>
                                    <element maxOccurs="unbounded" minOccurs="0"
name="HasContributorPersonQuery" type="query:PersonQueryType"/>
                                    <element maxOccurs="unbounded" minOccurs="0"
name="UsedOntologyEngineeringToolQuery" type="query_omv:OntologyEngineeringToolQueryType"/>
                                    <element maxOccurs="unbounded" minOccurs="0"
name="UsedOntologyEngineeringMethodologyQuery" type="query_omv:OntologyEngineeringMethodologyQueryType"/>
                                    <element maxOccurs="unbounded" minOccurs="0"
name="UsedKnowledgeRepresentationParadigmQuery" type="query_omv:KnowledgeRepresentationParadigmQueryType"/>
                                    <element maxOccurs="unbounded" minOccurs="0"
name="DesignedForOntologyTaskQuery" type="query_omv:OntologyTaskQueryType"/>
                                    <element maxOccurs="unbounded" minOccurs="0"
name="UseImportsQuery" type="query_omv:OntologyQueryType"/>
                                    <element maxOccurs="unbounded" minOccurs="0"
name="IsBackwardCompatibleWithQuery" type="query_omv:OntologyQueryType"/>
                                    <element maxOccurs="unbounded" minOccurs="0"
name="IsIncompatibleWithQuery" type="query_omv:OntologyQueryType"/>
                                </sequence>
                            </extension>
                        </complexContent>
                    </complexType>
                    <element name="OntologyQuery" type="query_omv:OntologyQueryType"/>
                </schema>
  </types>
  <message name="msgAdhocQueryResponse">
    <part name="partAdhocQueryResponse" element="query:AdhocQueryResponse">
    </part>
  </message>
  <message name="msgRemoveObjectsRequest">
    <part name="partRemoveObjectsRequest" element="lcm:RemoveObjectsRequest">
    </part>
  </message>
  <message name="msgUndeprecateObjectsRequest">
    <part name="partUndeprecateObjectsRequest" element="lcm:UndeprecateObjectsRequest">
    </part>
```

```
  </message>
  <message name="msgSubmitObjectsRequest">
    <part name="partSubmitObjectsRequest" element="lcm:SubmitObjectsRequest">
    </part>
  </message>
  <message name="msgRegistryResponse">
<documentation>Defines a RegistryResponse message.</documentation>
    <part name="partRegistryResponse" element="rs:RegistryResponse">
    </part>
  </message>
  <message name="msgAdhocQueryRequest">
    <part name="partAdhocQueryRequest" element="query:AdhocQueryRequest">
    </part>
  </message>
  <message name="msgUpdateObjectsRequest">
    <part name="partUpdateObjectsRequest" element="lcm:UpdateObjectsRequest">
    </part>
  </message>
  <message name="msgApproveObjectsRequest">
    <part name="partApproveObjectsRequest" element="lcm:ApproveObjectsRequest">
    </part>
  </message>
  <message name="msgDeprecateObjectsRequest">
    <part name="partDeprecateObjectsRequest" element="lcm:DeprecateObjectsRequest">
    </part>
  </message>
  <portType name="QueryManagerPortType">
    <operation name="submitAdhocQuery">
      <input message="tns:msgAdhocQueryRequest">
    </input>
      <output message="tns:msgAdhocQueryResponse">
    </output>
    </operation>
  </portType>
  <portType name="LifeCycleManagerPortType">
    <operation name="approveObjects">
      <input message="tns:msgApproveObjectsRequest">
    </input>
      <output message="tns:msgRegistryResponse">
    </output>
    </operation>
    <operation name="deprecateObjects">
      <input message="tns:msgDeprecateObjectsRequest">
    </input>
      <output message="tns:msgRegistryResponse">
    </output>
    </operation>
    <operation name="undeprecateObjects">
      <input message="tns:msgUndeprecateObjectsRequest">
    </input>
      <output message="tns:msgRegistryResponse">
    </output>
    </operation>
    <operation name="removeObjects">
      <input message="tns:msgRemoveObjectsRequest">
    </input>
      <output message="tns:msgRegistryResponse">
    </output>
    </operation>
    <operation name="submitObjects">
      <input message="tns:msgSubmitObjectsRequest">
    </input>
      <output message="tns:msgRegistryResponse">
    </output>
    </operation>
    <operation name="updateObjects">
      <input message="tns:msgUpdateObjectsRequest">
    </input>
      <output message="tns:msgRegistryResponse">
    </output>
    </operation>
  </portType>
  <binding name="LifeCycleManagerSOAPBinding" type="tns:LifeCycleManagerPortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="approveObjects">
```

```
    <soap:operation soapAction="urn:oasis:names:tc:ebxml-regrep:wsdl:registry:bindings:3.0:LifeCycleManagerPortType#approveObjects"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
  <operation name="deprecateObjects">
    <soap:operation soapAction="urn:oasis:names:tc:ebxml-regrep:wsdl:registry:bindings:3.0:LifeCycleManagerPortType#deprecateObjects"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
  <operation name="undeprecateObjects">
    <soap:operation soapAction="urn:oasis:names:tc:ebxml-regrep:wsdl:registry:bindings:3.0:LifeCycleManagerPortType#undeprecateObjects"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
  <operation name="removeObjects">
    <soap:operation soapAction="urn:oasis:names:tc:ebxml-regrep:wsdl:registry:bindings:3.0:LifeCycleManagerPortType#removeObjects"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
  <operation name="submitObjects">
    <soap:operation soapAction="urn:oasis:names:tc:ebxml-regrep:wsdl:registry:bindings:3.0:LifeCycleManagerPortType#submitObjects"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
  <operation name="updateObjects">
    <soap:operation soapAction="urn:oasis:names:tc:ebxml-regrep:wsdl:registry:bindings:3.0:LifeCycleManagerPortType#updateObjects"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
 </binding>
 <binding name="QueryManagerSOAPBinding" type="tns:QueryManagerPortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="submitAdhocQuery">
    <soap:operation soapAction="urn:oasis:names:tc:ebxml-regrep:wsdl:registry:bindings:3.0:QueryManagerPortType#submitAdhocQuery"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
 </binding>
 <service name="ebXMLRegistrySOAPService">
  <port name="QueryManagerPort" binding="tns:QueryManagerSOAPBinding">
    <soap:address location="http://10.22.21.64:8080/axis2/services/ebXMLRegistrySOAPService"/>
  </port>
  <port name="LifeCycleManagerPort" binding="tns:LifeCycleManagerSOAPBinding">
    <soap:address location="http://10.22.21.64:8080/axis2/services/ebXMLRegistrySOAPService"/>
  </port>
 </service>
</definitions>
```

6 Glossary

NeOn Registry Service	A web service which realizes the web service interface defined for accessing the ontology registry in NeOn. There may be multiple implementation of that web service interface and thus also multiple NeOn Registry Services.
NeOn Registry Service Oyster	The Oyster-based implementation of the NeOn registry service
NeOn Registry Service CentraSite	The CentraSite-based implementation of the NeOn registry service

7 References

Fuger2005	Fuger, S., Najmi, F., Stojanovic, N., ebXML Registry Information Model Version 3.0, OASIS, 2005 http://docs.oasis-open.org/regrep/v3.0/specs/regrep-rim-3.0-os.pdf
FugerNajmi2005	Fuger, S., Najmi, F., Stojanovic, N., ebXML Registry Services and Protocols Version 3.0, OASIS, 2005 http://docs.oasis-open.org/regrep/v3.0/specs/regrep-rs-3.0-os.pdf
Gomez2006	Gómez-Pérez J.M, Buil C, Muñoz, O. Requirements on NeOn architecture. NeOn Deliverable D6.1.1, 2006.
Najmi2002	Najmi, F., Java API for XML Registries (JAXR) Sun JCP 093, 2002 http://jcp.org/en/jsr/detail?id=93
Palma2007	Palma, R., Hartmann, Bontas, E., J., Haase, P., Technical Report on O M V - Ontology Metadata Vocabulary for the SemanticWeb, OMV consortium, 2007, http://www.omv.org
PalmaHaase2007	Palma R., Haase, P., Wang Y., D'Aquin M. Propagation Models and strategies. NeOn Deliverable D1.3.1, 2007
Wang2006	Wang, Y., Haase, P., Palma, R., Prototypes for Managing Networked Ontologies, NeOn Deliverable D1.4.1, 2006